Networking protocols and administration

ITV8030

lecture 3: udp, tcp

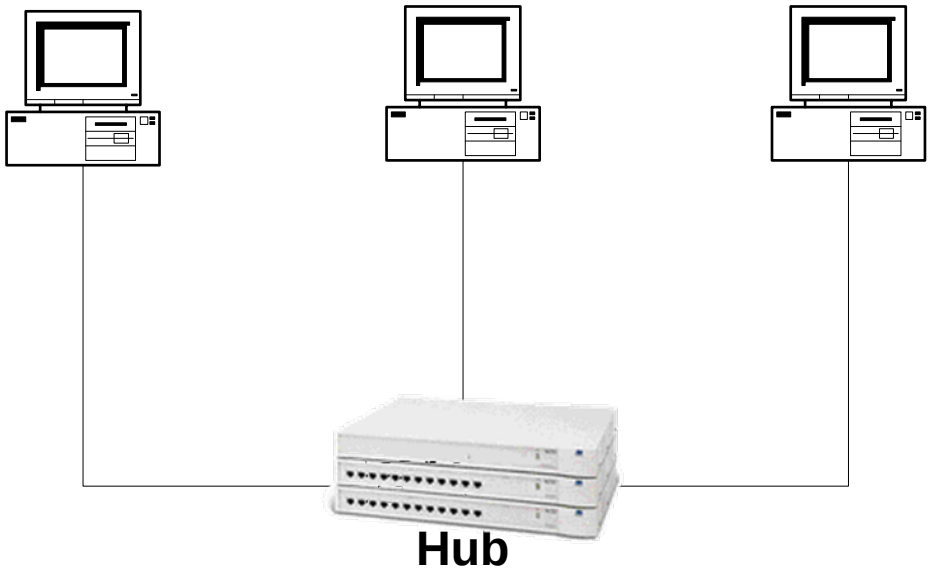
lecture plan

- Refresher: ethernet
- UDP protocol
- TCP protocol:
  - beginning and middle part of details

- Using mostly slides from Univ of Virginia / Univ of Toronto

Current Star Topology

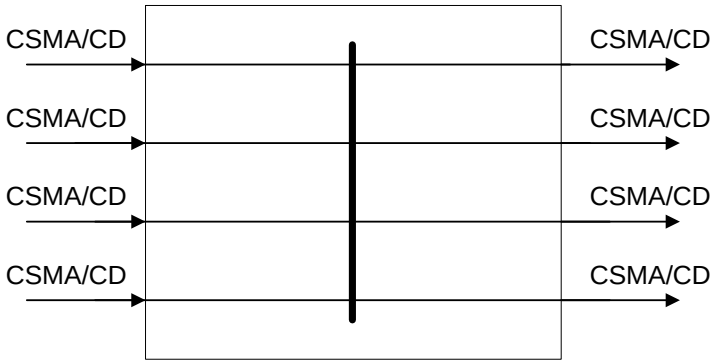
- Starting with 10Base-T, stations are connected to a hub in a star configuration



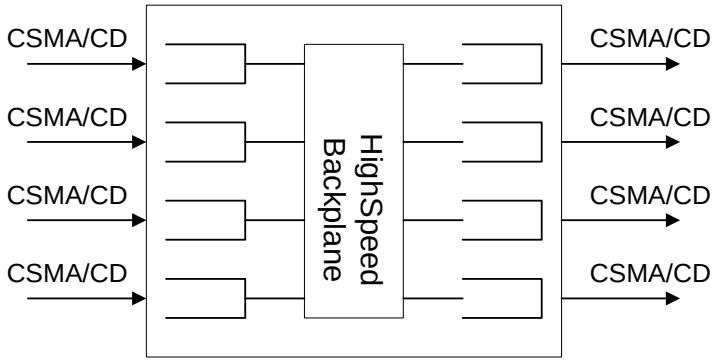
Ethernet Hubs vs. Ethernet Switches

- An **Ethernet switch** is a packet switch for Ethernet frames
  - Buffering of frames prevents collisions.
  - Each port is isolated and builds its own collision domain
- An **Ethernet Hub** does not perform buffering:
  - Collisions occur if two frames arrive at the same time.

Hub



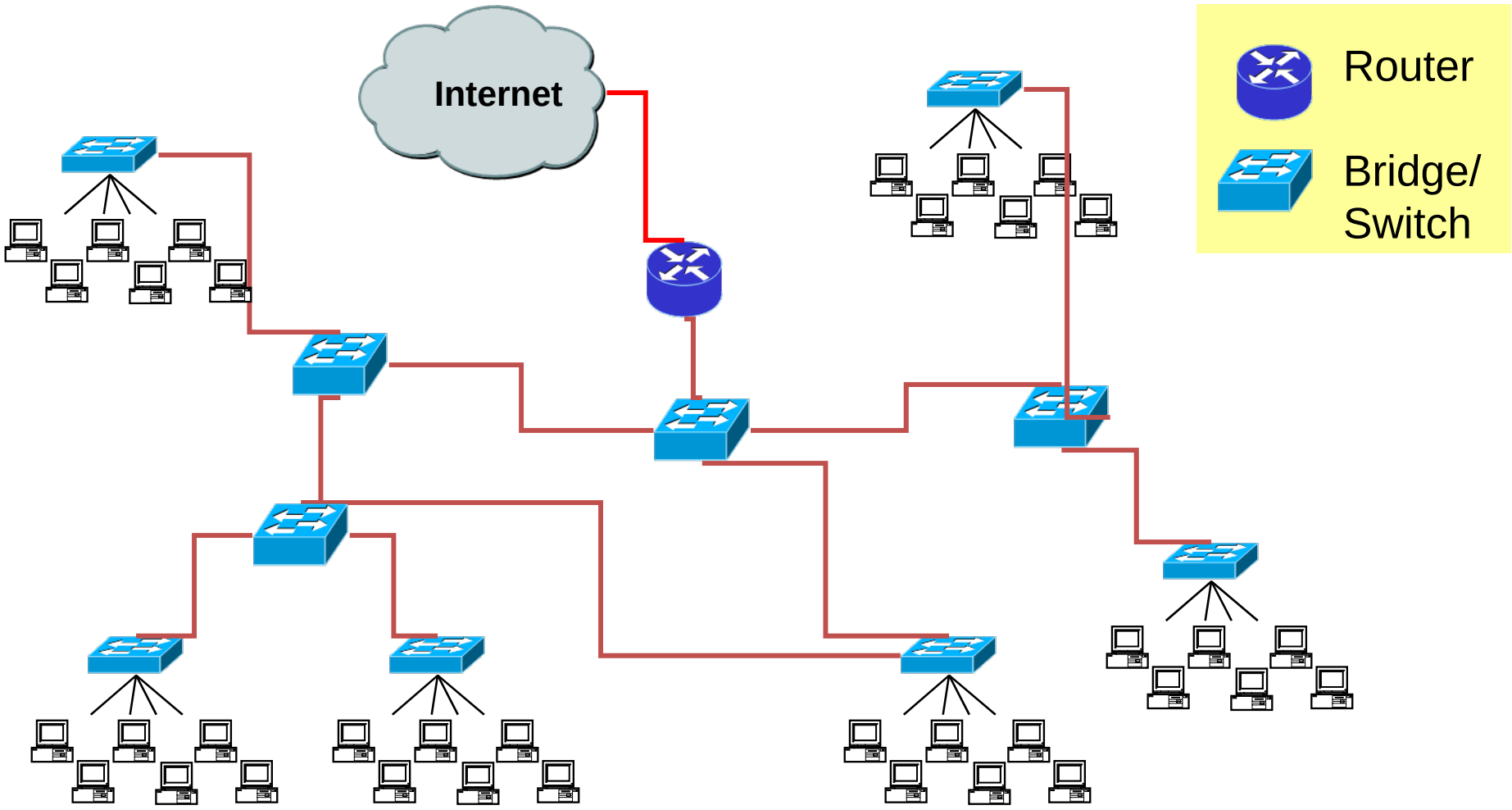
Switch



Input Buffers

Output Buffers

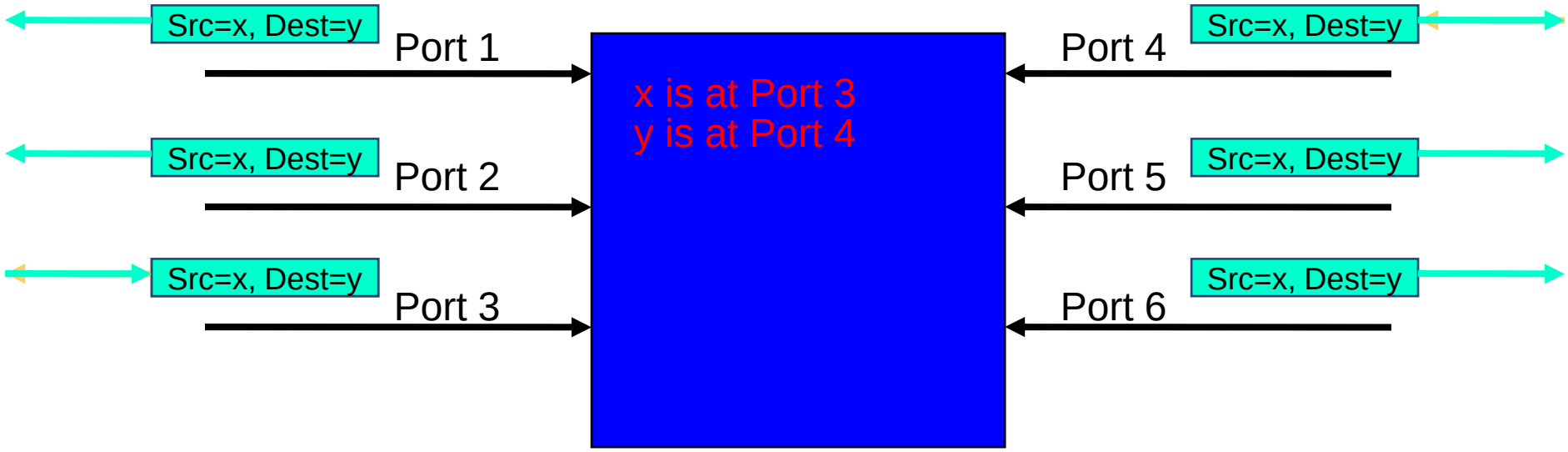
A Switched Enterprise Network



(2) Address Learning (Learning Bridges)

- Routing tables entries are set automatically with a simple heuristic:

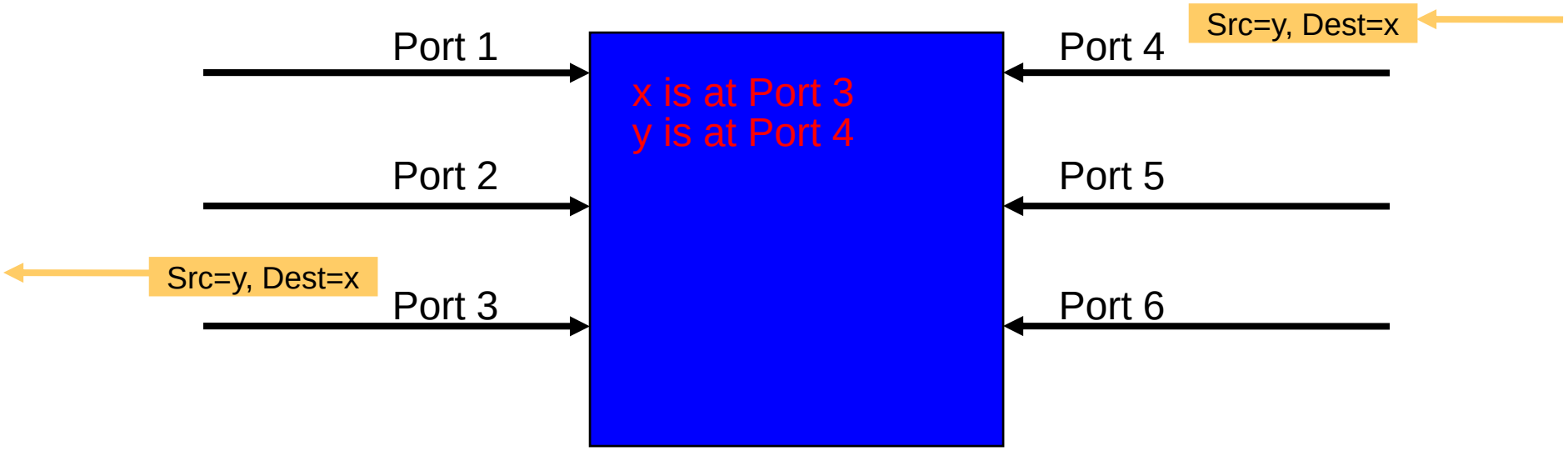
  The source field of a frame that arrives on a port tells which hosts are reachable from this port.



(2) Address Learning (Learning Bridges)

Learning Algorithm:

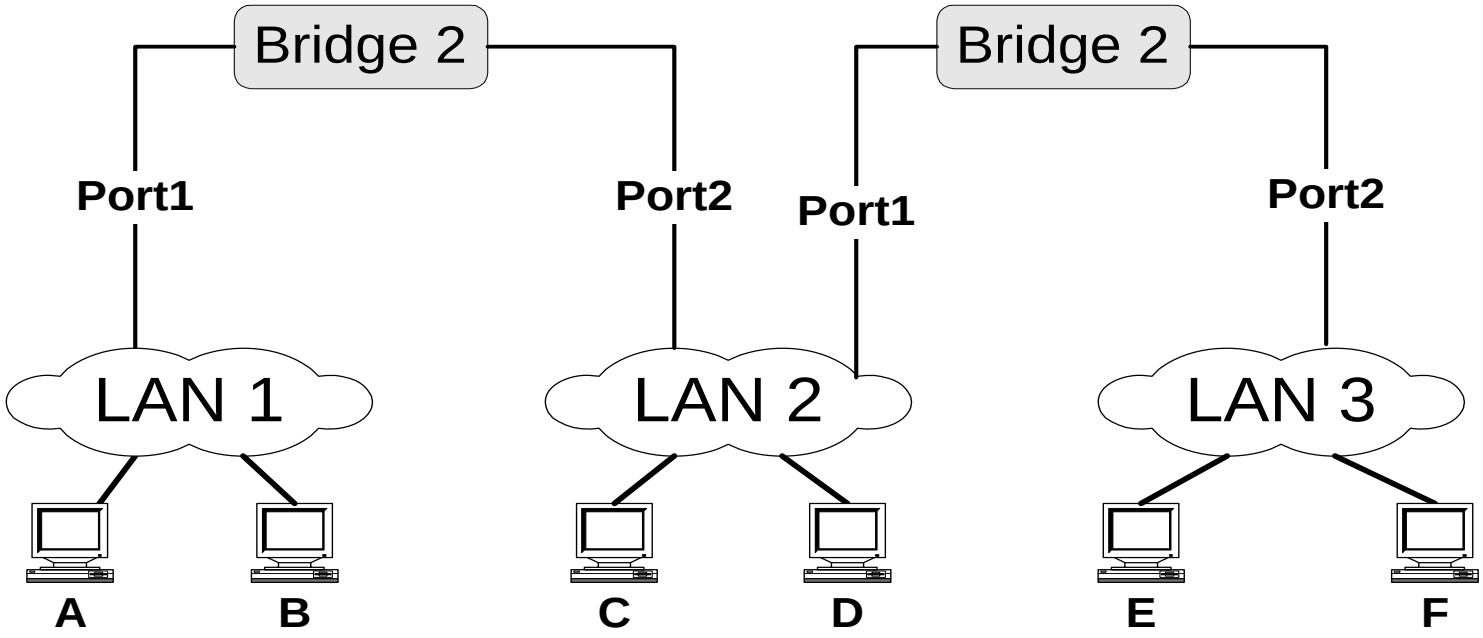
- For each frame received, the source stores the source field in the forwarding database together with the port where the frame was received.
- All entries are deleted after some time (default is 15 seconds).



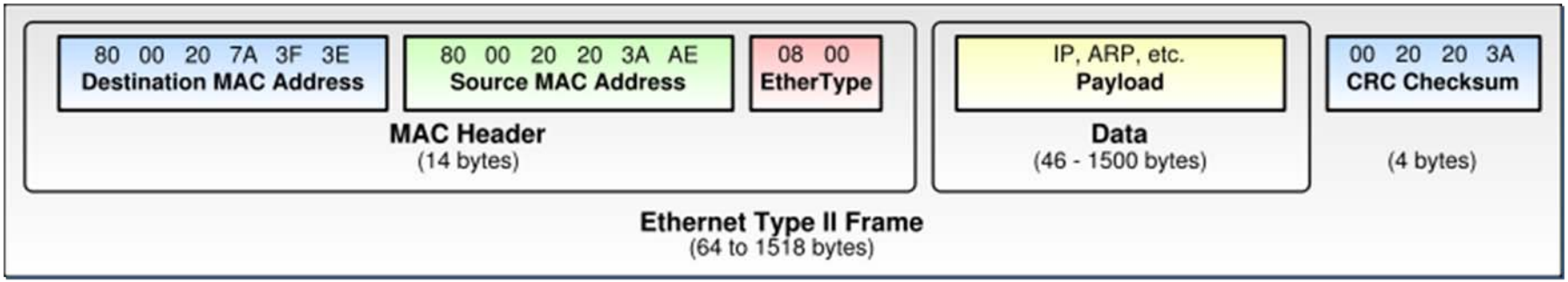
Example

Consider the following packets:
(Src=A, Dest=F), (Src=C, Dest=A), (Src=E, Dest=C)

What have the bridges learned?



Ethernet frame format



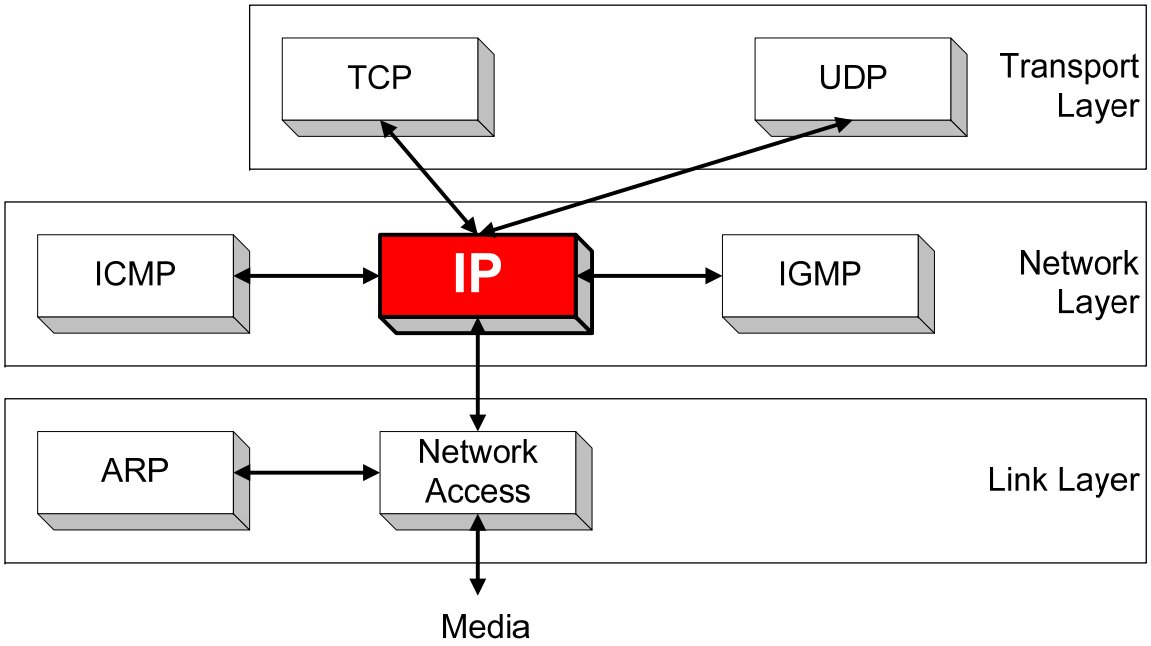
- Preamble 7 octets of 10101010
- Start-of-Frame-Delimiter 1 octet of 10101011

- **MAC destination 6 octets**
- **MAC source 6 octets**
- **Ethertype/Length 2 octets**
- **Payload 46-1500 octets**
- **CRC32 4 octets**

- (Postamble 1 octet of 01111110)
- Interframe gap 960 ns (100M)

Orientation

- IP (Internet Protocol) is a Network Layer Protocol.

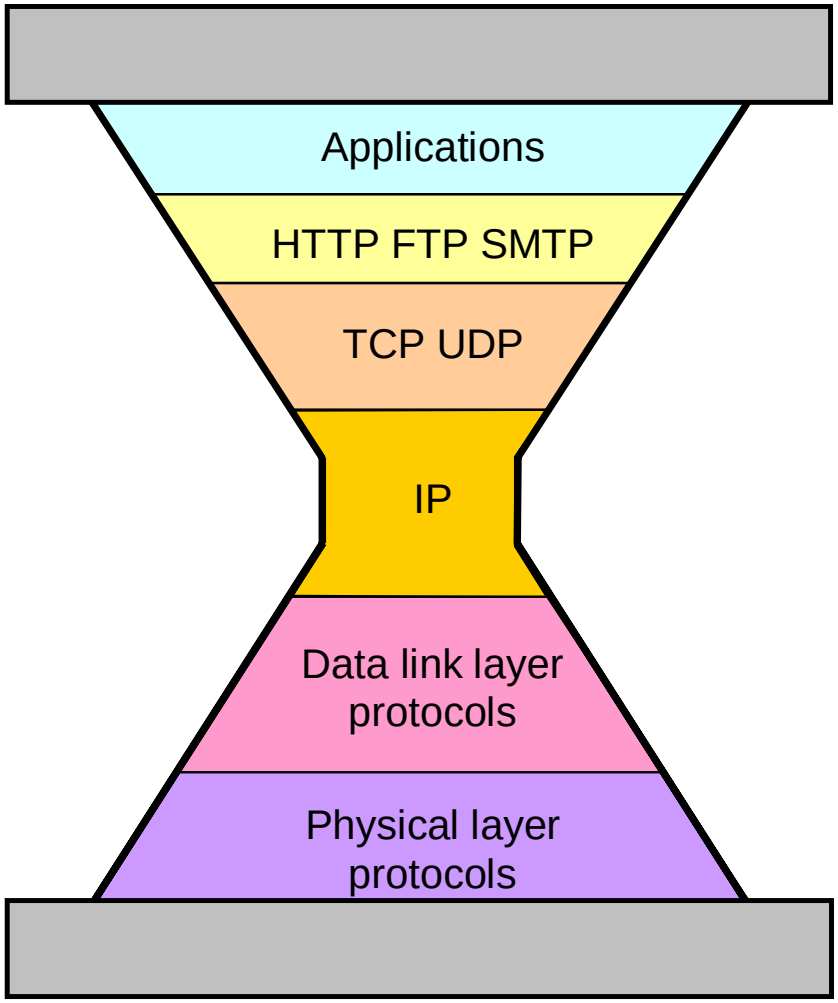


IP: The waist of the hourglass

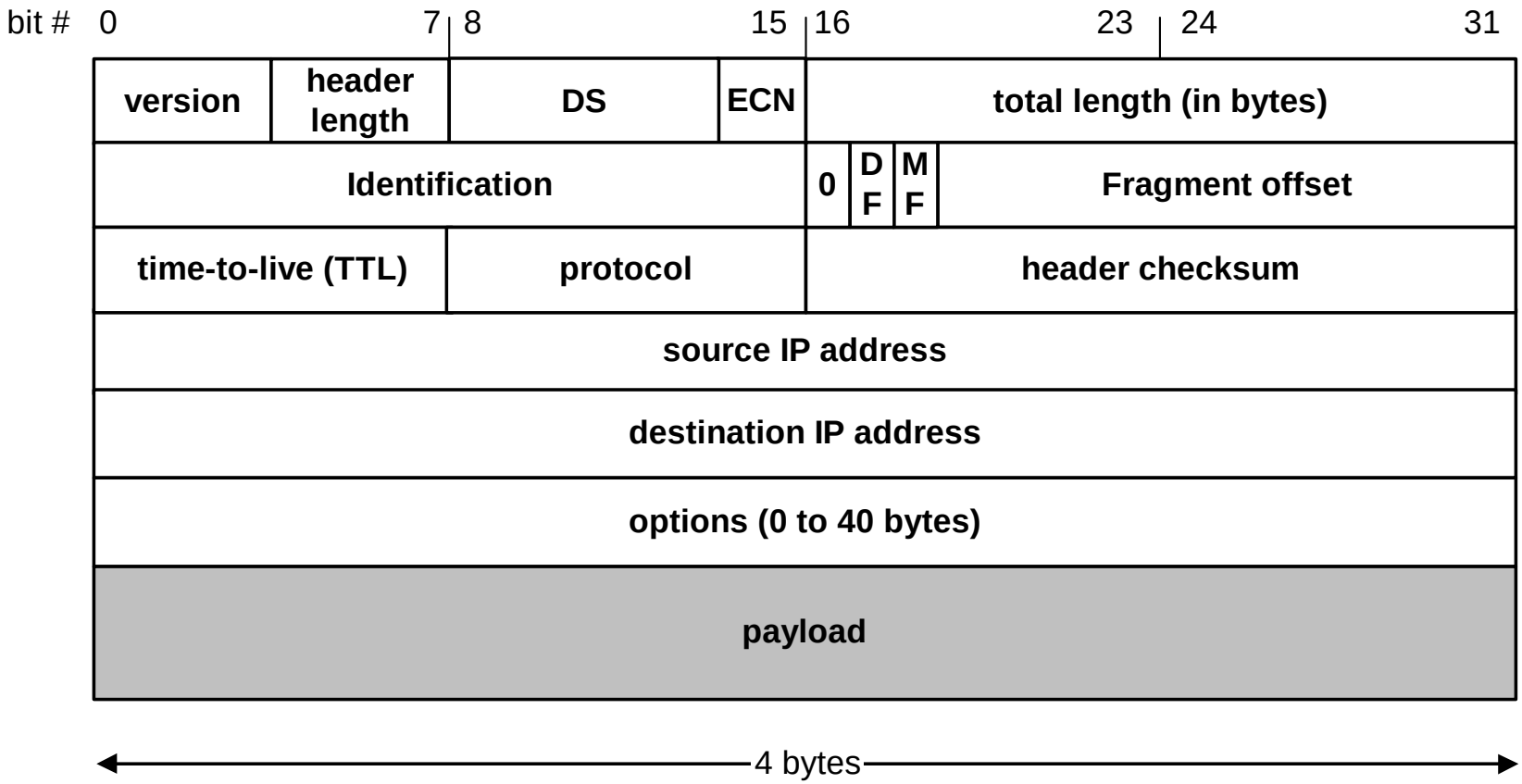
- **IP is the waist of the hourglass of the Internet protocol architecture**

- Multiple higher-layer protocols
- Multiple lower-layer protocols

- Only one protocol at the network layer.



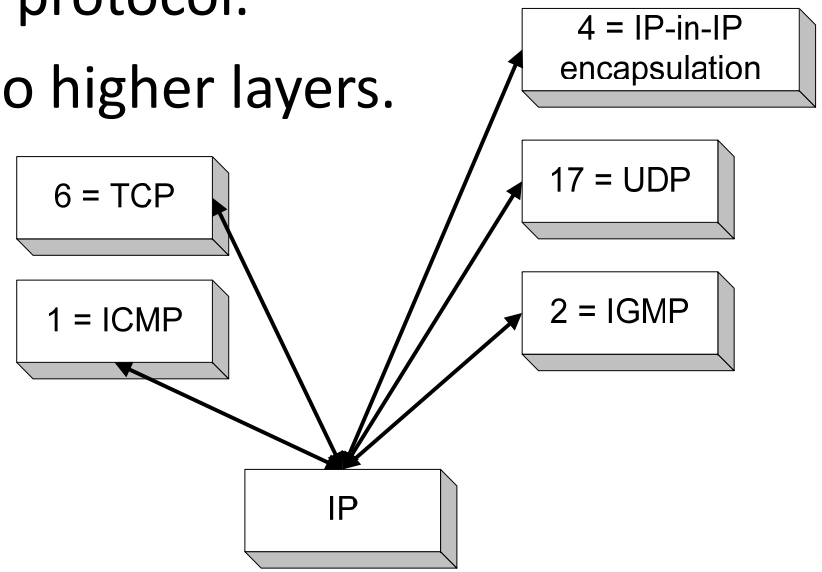
IP Datagram Format



- 20 bytes ≤ Header Size < 2^4 x 4 bytes = 60 bytes
- 20 bytes ≤ Total Length < 2^{16} bytes = 65536 bytes

Fields of the IP Header

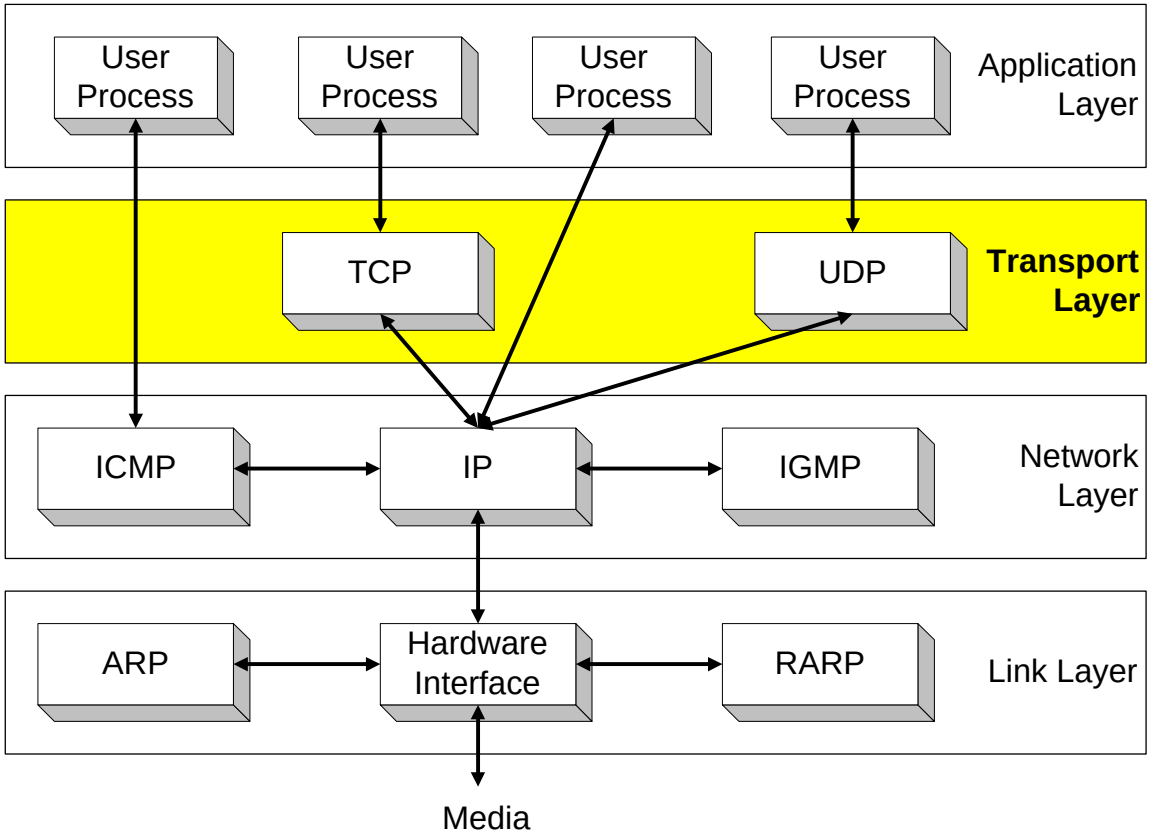
- **Protocol (1 byte):**
  - Specifies the higher-layer protocol.
  - Used for demultiplexing to higher layers.



- **Header checksum (2 bytes):** A simple 16-bit long checksum which is computed for the header of the datagram.

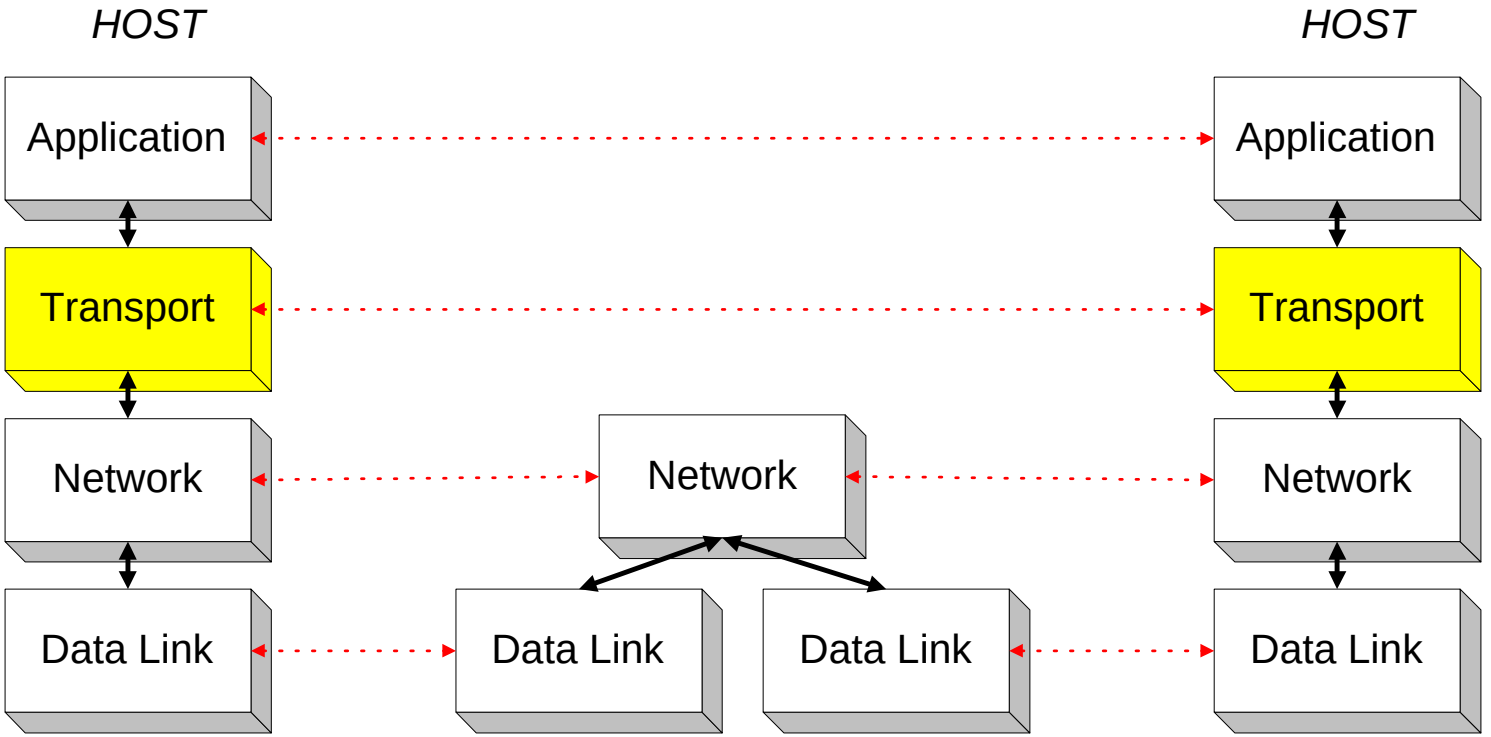
Orientation

- We move one layer up and look at the transport layer.



Orientation

- Transport layer protocols are end-to-end protocols
- They are only implemented at the hosts



Transport Protocols in the Internet

UDP - User Datagram Protocol

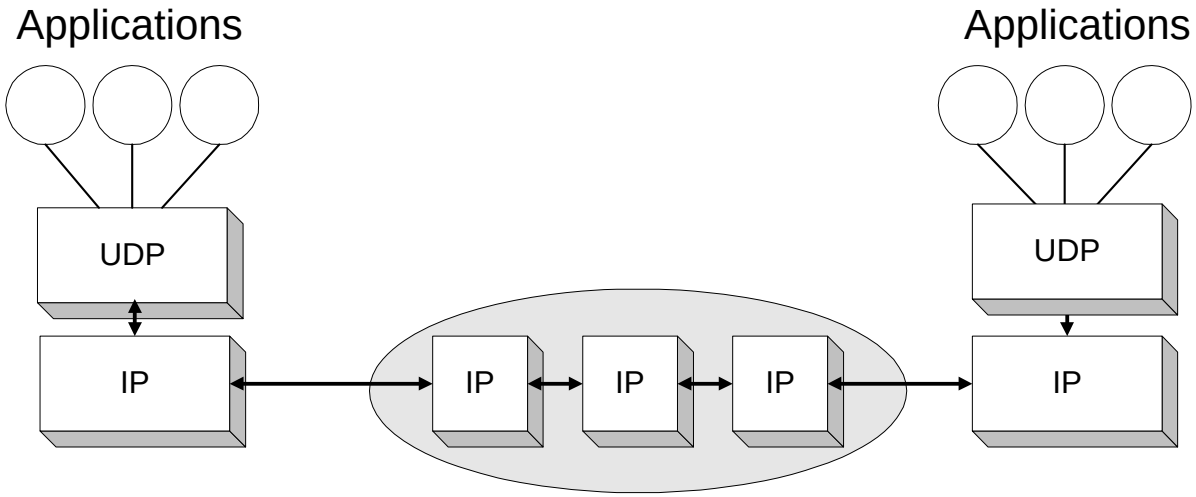
- datagram oriented
- unreliable, connectionless
- simple
- unicast and multicast
- useful only for few applications, e.g., multimedia applications
- used a lot for services
  - network management (SNMP), routing (RIP), naming (DNS), etc.

TCP - Transmission Control Protocol

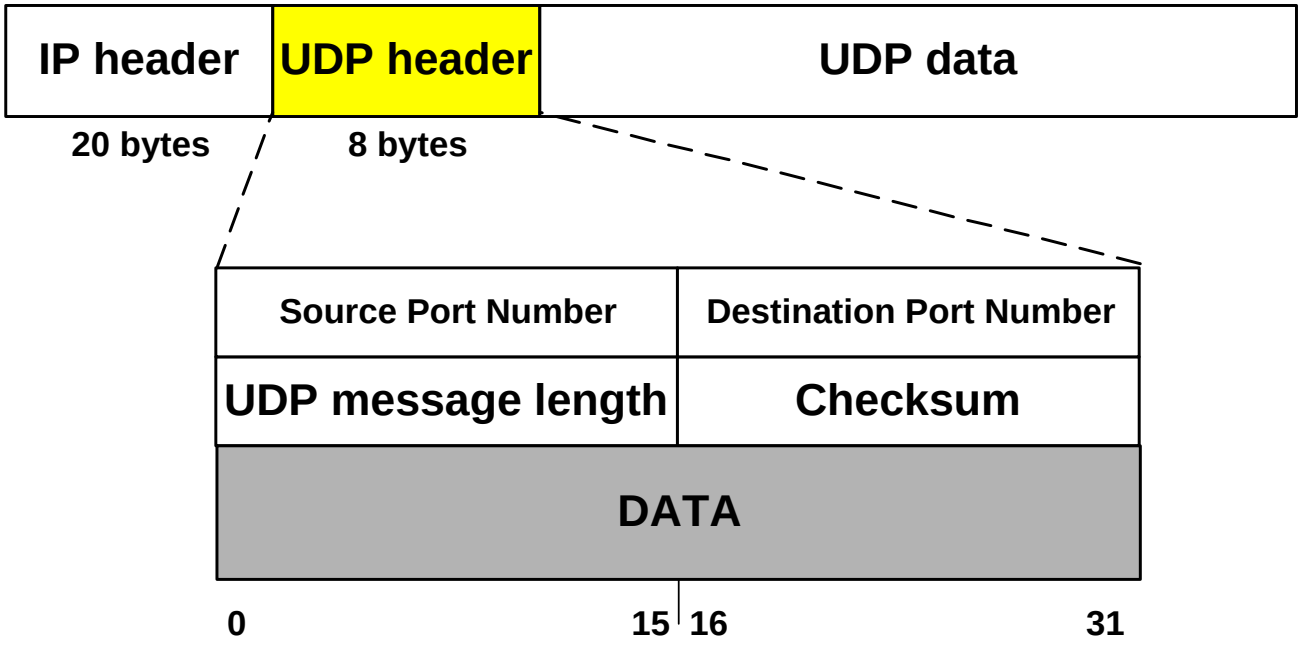
- stream oriented
- reliable, connection-oriented
- complex
- only unicast
- used for most Internet applications:
  - web (http), email (smtp), file transfer (ftp), terminal (telnet), etc.

UDP - User Datagram Protocol

- UDP is supports unreliable transmissions of datagrams
- UDP merely extends the host-to-to-host delivery service of IP datagram to an application-to-application service
- The only thing that UDP adds is multiplexing and demultiplexing

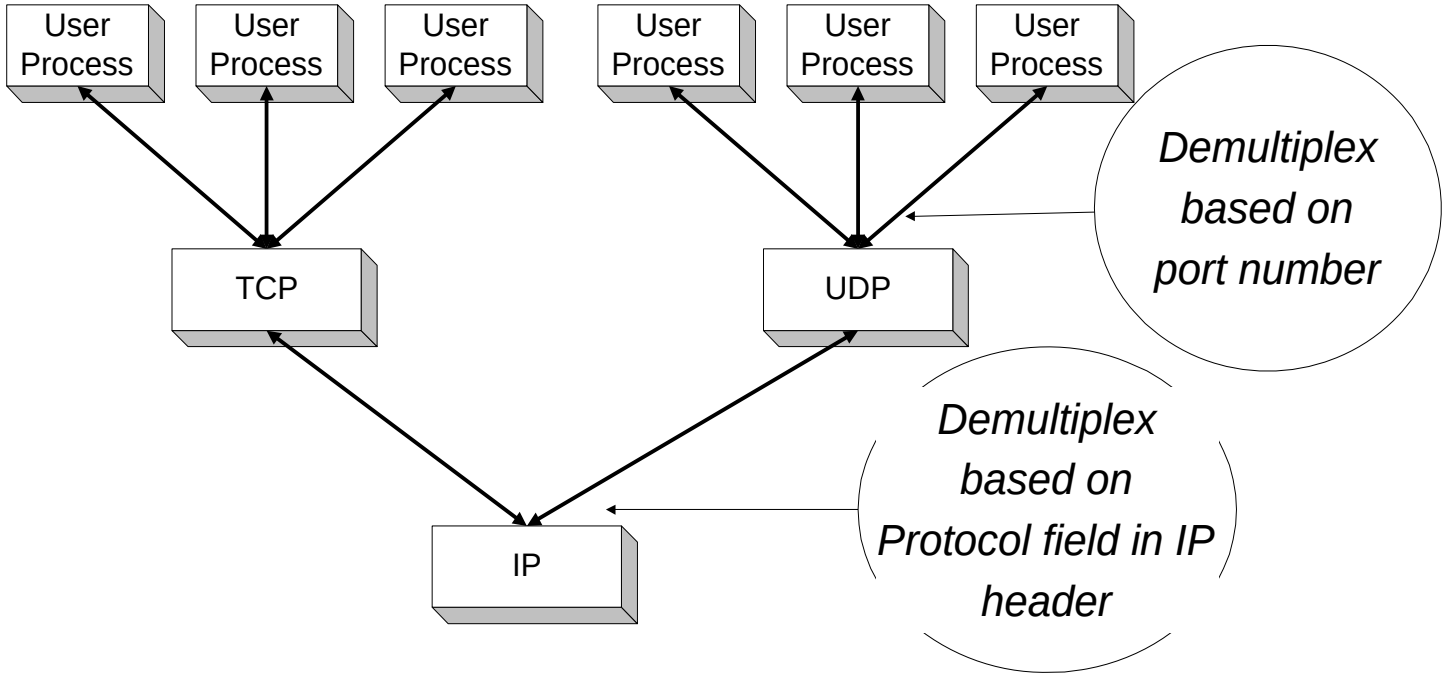


UDP Format


IP header
UDP header
UDP data
20 bytes
8 bytes
Source Port Number
Destination Port Number
UDP message length
Checksum
DATA
0
15
16
31

Port Numbers

- UDP (and TCP) use port numbers to identify applications
- A globally unique address at the transport layer (for both UDP and TCP) is a tuple **<IP address, port number>**
- There are 65,535 UDP ports per host.



Port numbers

- The **Well Known Ports** are those in the range 0–1023. On Unix-like operating systems, opening a port in this range to receive incoming connections requires administrative privileges or possessing of *CAP_NET_BIND_SERVICE* capability.
- The **Registered Ports** are those in the range 1024–49151.
- The **Dynamic and/or Private Ports** are those in the range 49152–65535. Randomly chosen port numbers out of this range are called ephemeral ports. These ports are not permanently assigned to any publicly defined application.
- http://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers

UDP client example

- In python:

```
import socket
SERVER_ADDRESS = '127.0.0.1'
SERVER_PORT = 10000
client = socket.socket(socket.AF_INET,
  socket.SOCK_DGRAM, socket.IPPROTO_UDP)
for i in range(3):
        print 'Sending packet %d' % i
        message = 'This is packet %d' % i
        client.sendto(message, (SERVER_ADDRESS,
  SERVER_PORT))
client.close()
```

UDP server example

- In python:

```
import socket
PORT = 10000
BUFLEN = 512
server = socket.socket(socket.AF_INET,
  socket.SOCK_DGRAM, socket.IPPROTO_UDP)
server.bind(('', PORT))
while True:
        (message, address) = server.recvfrom(BUFLEN)
        print 'Received packet from %s:%d' %
  (address[0], address[1])
        print 'Data: %s' % message
```

VOIP

- Voice over IP (VoIP): Transmission of voice over Internet
- How VoIP works
  - Continuously sample audio
  - Convert each sample to digital form
  - Send digitized stream across Internet in packets
  - Convert the stream back to analog for playback
- Why VoIP
  - IP telephony is economic; High costs for traditional telephone switching equipments.

VOIP

- Challenge
  - Voice transmission delay
  - Call setup: call establishment, call termination, etc.
  - Backward compatibility with existing PSTN (Public Switched Telephone Network)

- IP Telephony Standards:
  - ITU (International Telecommunication Union) controls telephony standards.
  - IETF (Internet Engineering Task Force) controls TCP/IP standards.

VOIP Encoding, Transmission, ...

- Both groups agree on the basics for encoding and transmission of audio:
  - Audio is encoded using a well-known standard such as *Pulse Code Modulation* (PCM).
  - Audio is transferred using the *Real-time Transport Protocol* (RTP).
  - RTP message is encapsulated in a UDP datagram that is further encapsulated in an IP datagram for transmission.

VOIP Encoding, Transmission, ...

- UDP is used for transport because
  - lower overhead: audio must be played as it arrives.
  - Playback cannot be stopped to wait for a retransmitted packet.

- Two independent RTP sessions exist, because an IP phone call involves transfer in two directions
  - IP phone acts as sender for outgoing data, and
  - IP phone acts as receiver for incoming data.

VOIP signaling Systems & Protocols

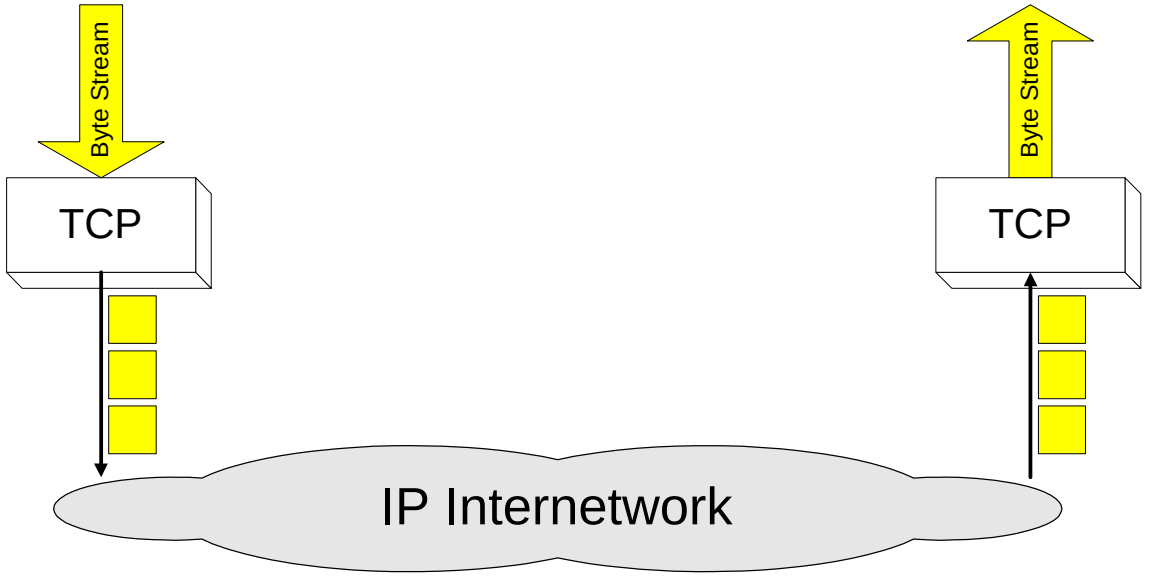
- Main complexity of VoIP: Call setup and call management.

- The process of establishing and terminating a call is called *Signaling*.
  - In traditional telephone system, signaling protocol is *SS7 (signaling System 7).*
  - In VoIP, signaling protocols are:
    - SIP (Session Initiation Protocol), by IETF
    - H.323, by ITU
    - Megaco & MGCP, jointly by IETF and IUT.
    - ....
  - VoIP signaling protocols should be able to interact with SS7.

TCP overview

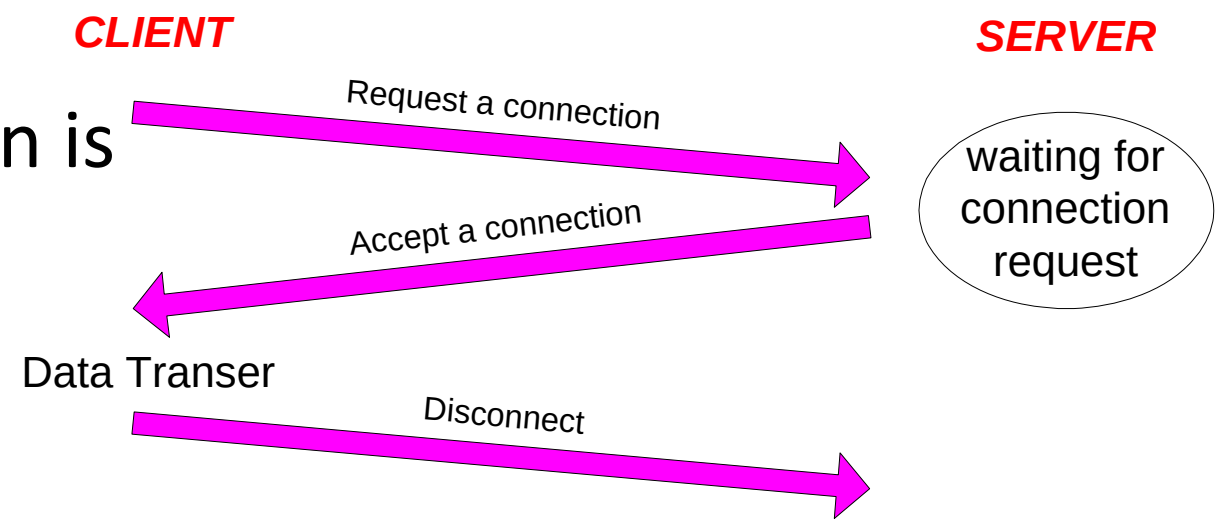
TCP = Transmission Control Protocol

- Connection-oriented protocol
- Provides a reliable unicast end-to-end byte stream over an unreliable internetwork.



Connection-Oriented

- Before any data transfer, TCP establishes a **connection**:
  - One TCP entity is waiting for a connection ("**server**")
  - The other TCP entity ("**client**") contacts the server
- The actual procedure for setting up connections is more complex.
- Each connection is full duplex



Reliable

Byte stream is broken up into chunks which are called **segments**

Receiver sends acknowledgements (ACKs) for segments

TCP maintains a timer. If an ACK is not received in time, the segment is retransmitted

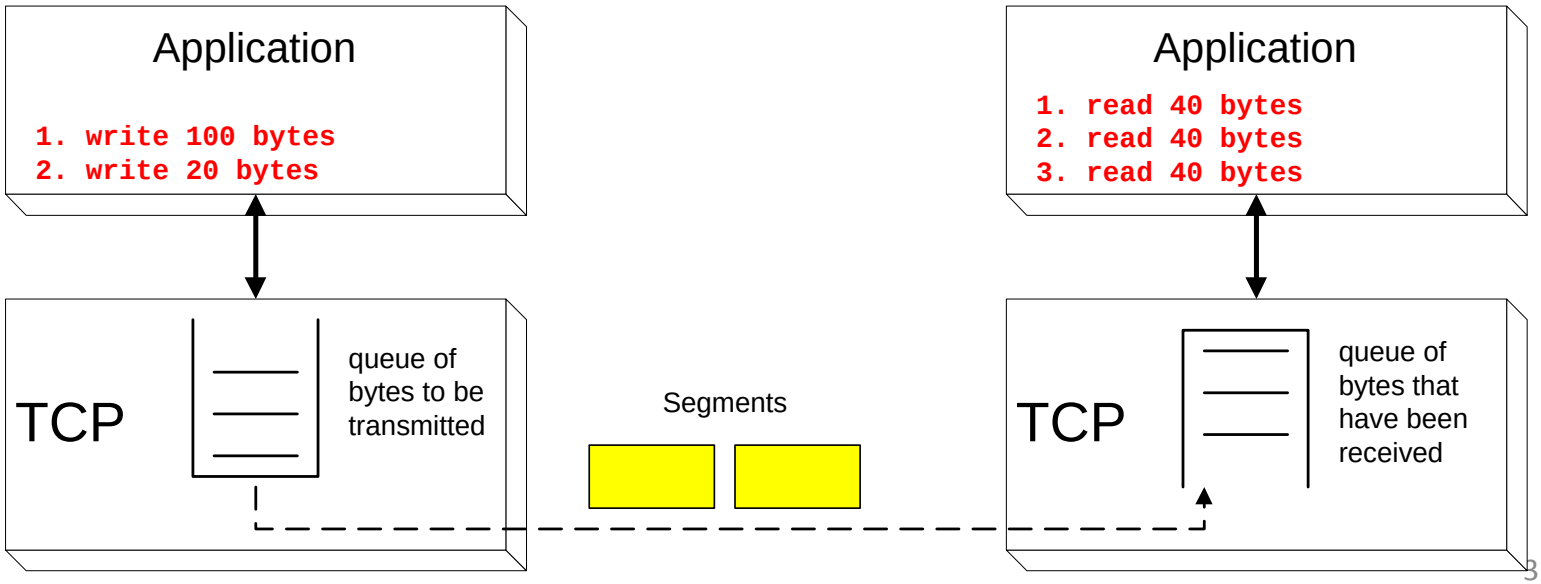
Detecting errors:

TCP has checksums for header and data. Segments with invalid checksums are discarded

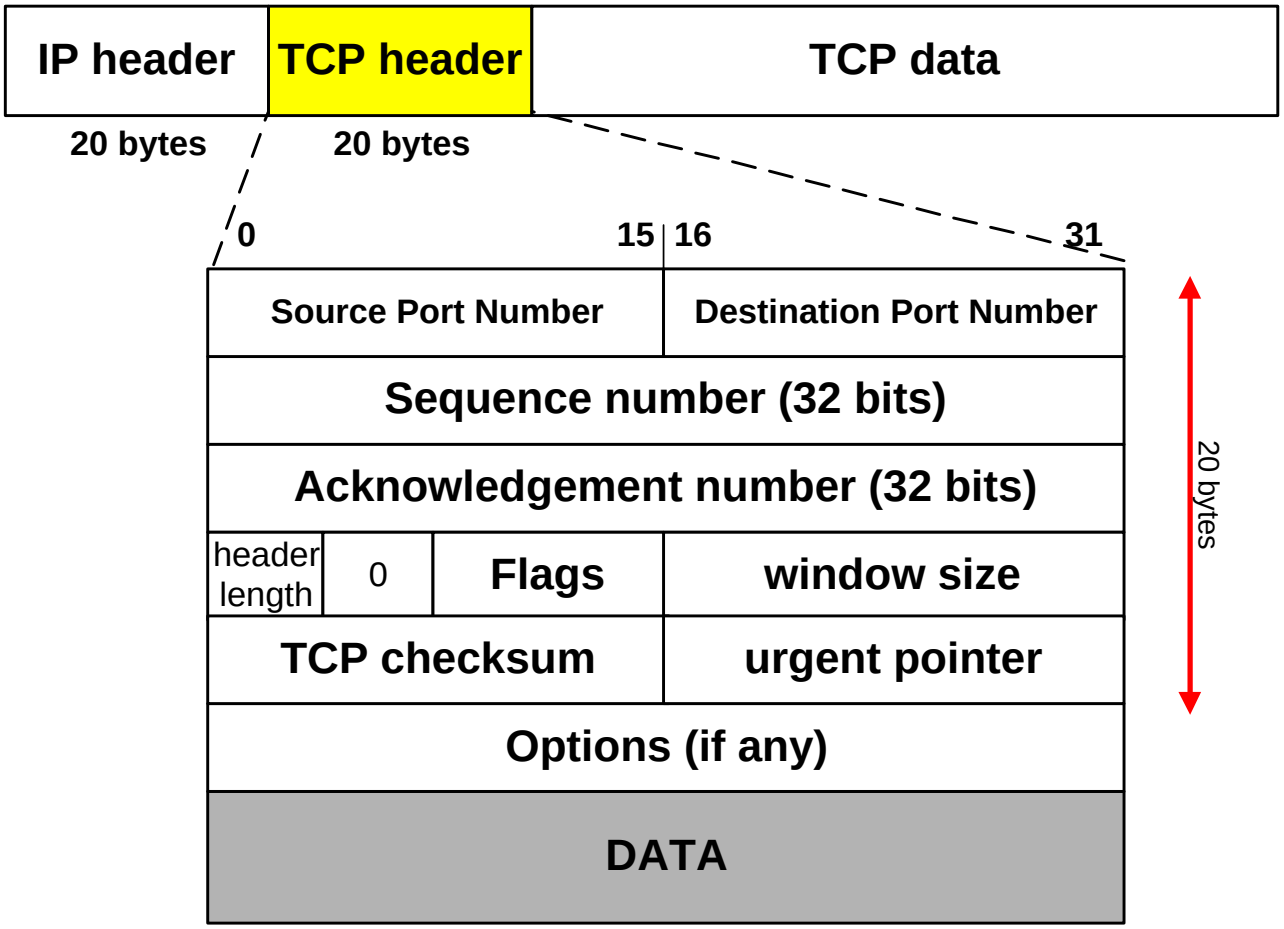
Each byte that is transmitted has a sequence number

Byte Stream Service

- To the lower layers, TCP handles data in blocks, the segments.
- To the higher layers TCP handles data as a sequence of bytes and does not identify boundaries between bytes
- So: Higher layers do not know about the beginning and end of segments !



TCP Format


IP header
TCP header
TCP data
20 bytes
20 bytes
0
15
16
31
Source Port Number
Destination Port Number
Sequence number (32 bits)
Acknowledgement number (32 bits)
header length
0
Flags
window size
TCP checksum
urgent pointer
Options (if any)
DATA
20 bytes

How to understand TCP

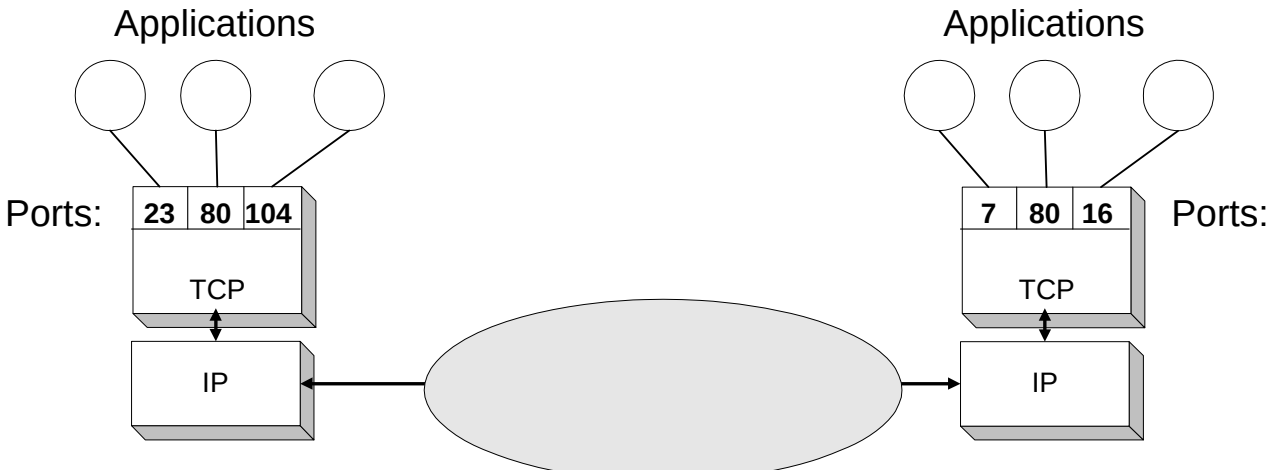
- Think like this:
  - **Core TCP**: protocol parts that must be implemented
  - **Optional** TCP protocol parts:
    - some implemented in sender/receiver, some not
    - sender and receiver advertise available options
    - important ones normally implemented
  - **Timing, packet size, etc options**:
    - main goal: better bandwidth usage, speedier operation
    - sender may determine at will
    - sender algorithms may vary

TCP vs UDP

- TCP does channel optimisations, "nice use"
- UDP is very rough and just sends packets
- http and voice/video on same channel
- UDP voice/video is normally stronger: TCP gives way to UDP

TCP header fields

- **Port Number:**
  - A port number identifies the endpoint of a connection.
  - A pair **<IP address, port number>** identifies one endpoint of a connection.
  - Two pairs **<client IP address, server port number>** and **<server IP address, server port number>** identify a TCP connection.



TCP header fields

- **Sequence Number (SeqNo):**
  - Sequence number is 32 bits long.
  - So the range of SeqNo is

    $0 <= SeqNo <= 2^{32} - 1 \approx 4.3$ Gbyte
  - Each sequence number identifies a byte in the byte stream
  - Initial Sequence Number (ISN) of a connection is set during connection establishment

TCP header fields

- **Acknowledgement Number (AckNo):**
  - Acknowledgements are piggybacked, I.e

    a segment from A -> B can contain an acknowledgement for a data sent in the B -> A direction
  - A host uses the AckNo field to send acknowledgements. (If a host sends an AckNo in a segment it sets the "**ACK flag**")
  - The AckNo contains the next SeqNo that a hosts wants to receive

    Example: The acknowledgement for a segment with sequence numbers 0-1500 is AckNo=1501

TCP header fields

- **Acknowledge Number (cont'd)**
  - TCP uses the sliding window flow protocol (see CS 457) to regulate the flow of traffic from sender to receiver
  - TCP uses the following variation of sliding window:
    - no NACKs (**N**egative **ACK**nowledgement)
    - only cumulative ACKs
- Example:

  Assume: Sender sends two segments with "1..1500" and "1501..3000", but receiver only gets the second segment.

  In this case, the receiver cannot acknowledge the second packet. It can only send AckNo=1

TCP header fields

- **Header Length (4bits):**
  - Length of header in 32-bit words
  - Note that TCP header has variable length (with minimum 20 bytes)

TCP header fields

- **Flag bits:**
  - **URG: Urgent pointer is valid**
    - If the bit is set, the following bytes contain an urgent message in the range:

      SeqNo <= urgent message <= SeqNo+urgent pointer
  - **ACK: Acknowledgement Number is valid**
  - **PSH: PUSH Flag**
    - Notification from sender to the receiver that the receiver should pass all data that it has to the application.
    - Normally set by sender when the sender's buffer is empty

TCP header fields

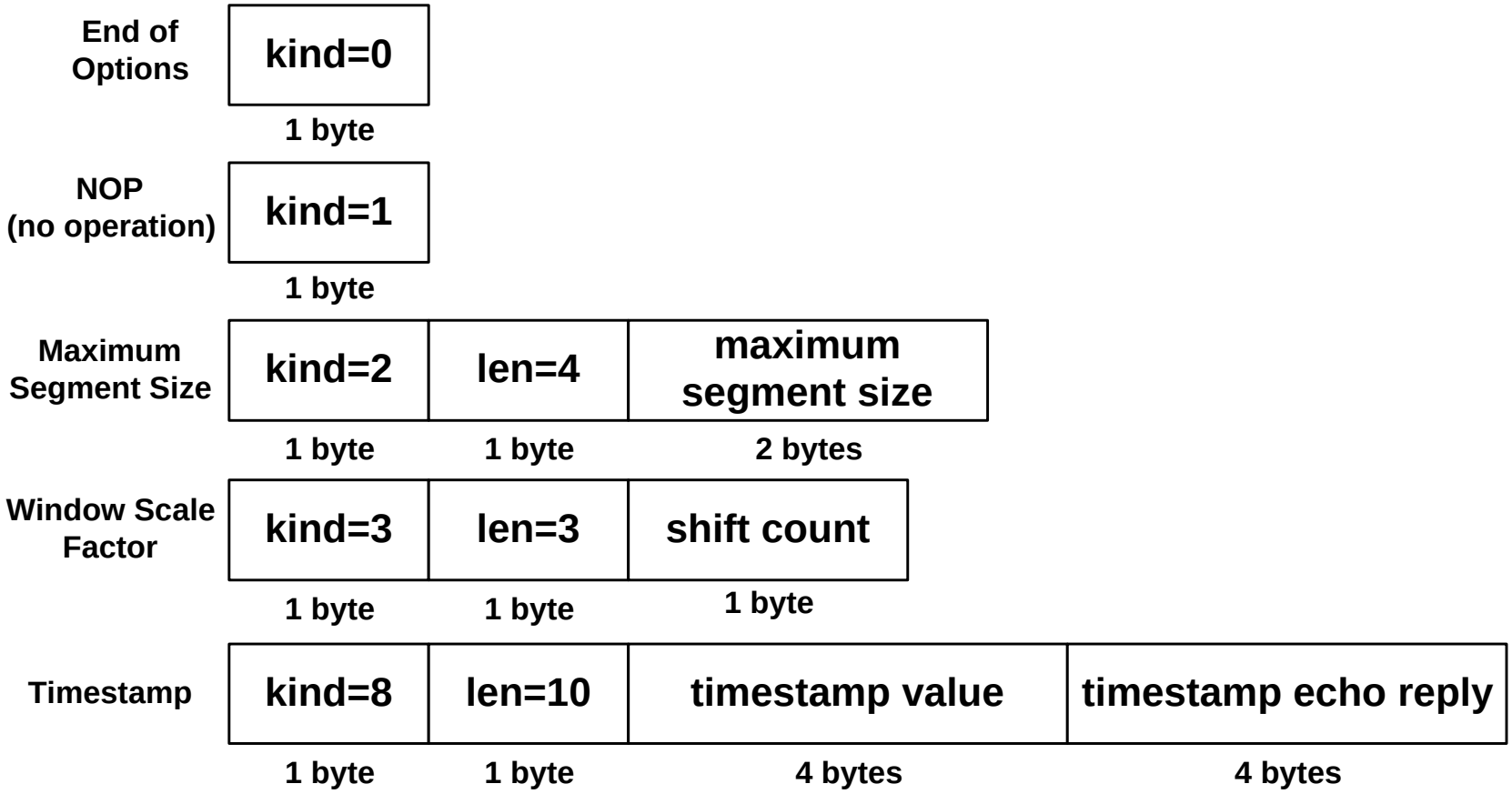
- **Flag bits:**
  - **RST: Reset the connection**
    - The flag causes the receiver to reset the connection
    - Receiver of a RST terminates the connection and indicates higher layer application about the reset
  - **SYN: Synchronize sequence numbers**
    - Sent in the first packet when initiating a connection
  - **FIN: Sender is finished with sending**
    - Used for closing a connection
    - Both sides of a connection must send a **FIN**

TCP header fields

- **Window Size:**
  - Each side of the connection advertises the window size
  - Window size is the maximum number of bytes that a receiver can accept.
  - Maximum window size is 2^{16}-1= 65535 bytes
- **TCP Checksum:**
  - TCP checksum covers over both TCP header **and** TCP data (also covers some parts of the IP header)
- **Urgent Pointer:**
  - Only valid if **URG** flag is set

TCP header fields

- **Options**:



TCP header fields

- **Options**:
  - **NOP** is used to pad TCP header to multiples of 4 bytes
  - **Maximum Segment Size**
  - **Window Scale Options**
    - » Increases the TCP window from 16 to 32 bits, I.e., the window size is interpreted differently
    - » This option can only be used in the SYN segment (first segment) during connection establishment time
  - **Timestamp Option**
    - » Can be used for roundtrip measurements

Connection Management in TCP

- **Opening a TCP Connection**
- **Closing a TCP Connection**
- **Special Scenarios**
- **State Diagram**

TCP Connection Establishment

- TCP uses a **three-way handshake** to open a connection:

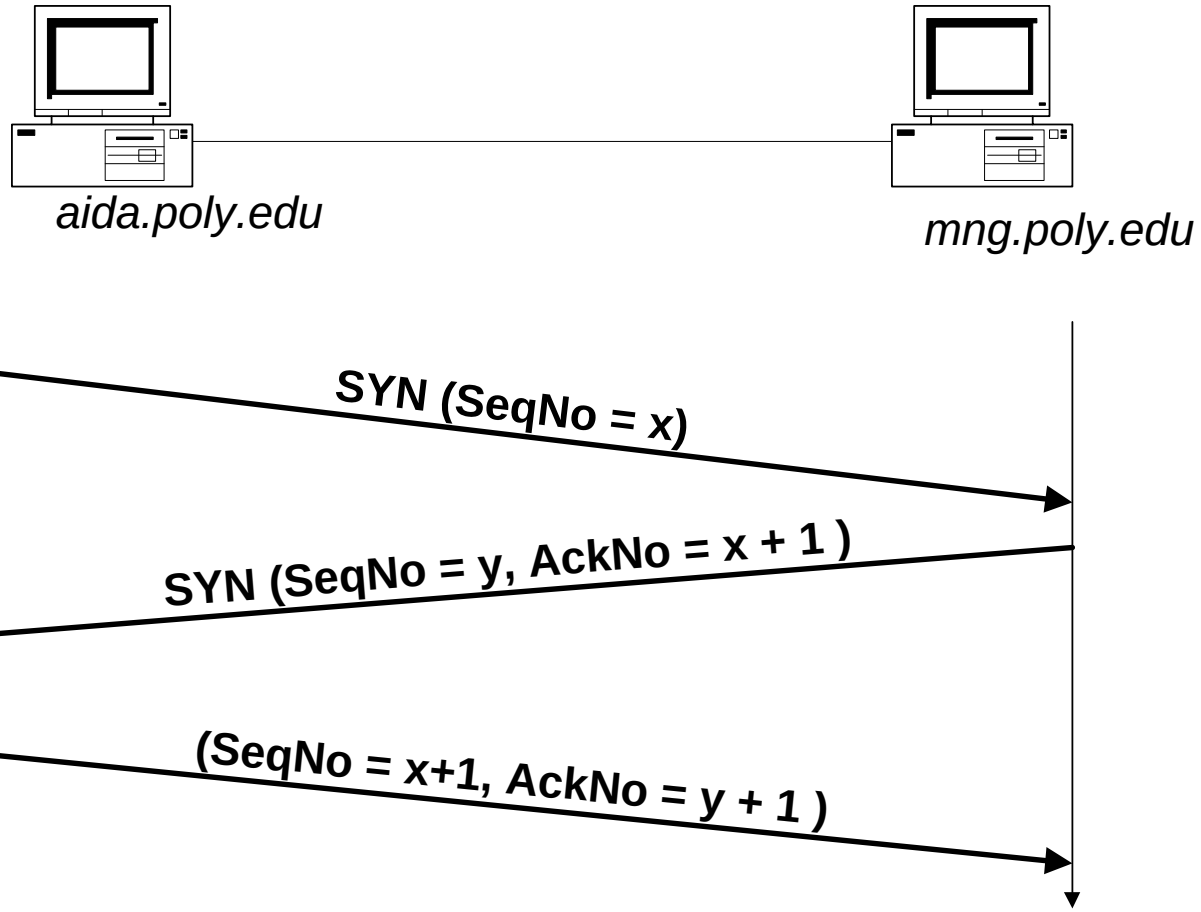
  (1) ACTIVE OPEN: Client sends a segment with
  - SYN bit set *
  - port number of client
  - initial sequence number (ISN) of client

  (2) PASSIVE OPEN: Server responds with a segment with
  - SYN bit set *
  - initial sequence number of server
  - ACK for ISN of client

  (3) Client acknowledges by sending a segment with:
  - ACK ISN of server

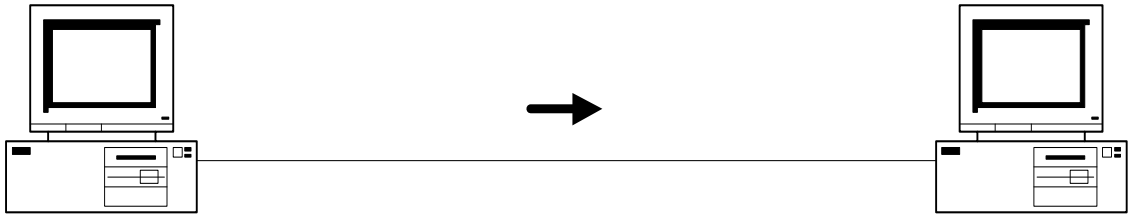
  * counts as one byte

Three-Way Handshake



A Closer Look with tcpdump

aida issues
an "telnet mng"



aida.poly.edu *mng.poly.edu*

1 aida.poly.edu.1121 > mng.poly.edu.telnet: S 1031880193:1031880193(0)
win 16384 <mss 1460,nop,wscale 0,nop,nop,timestamp>

2 mng.poly.edu.telnet > aida.poly.edu.1121: S 172488586:172488586(0)
ack 1031880194 win 8760 <mss 1460>

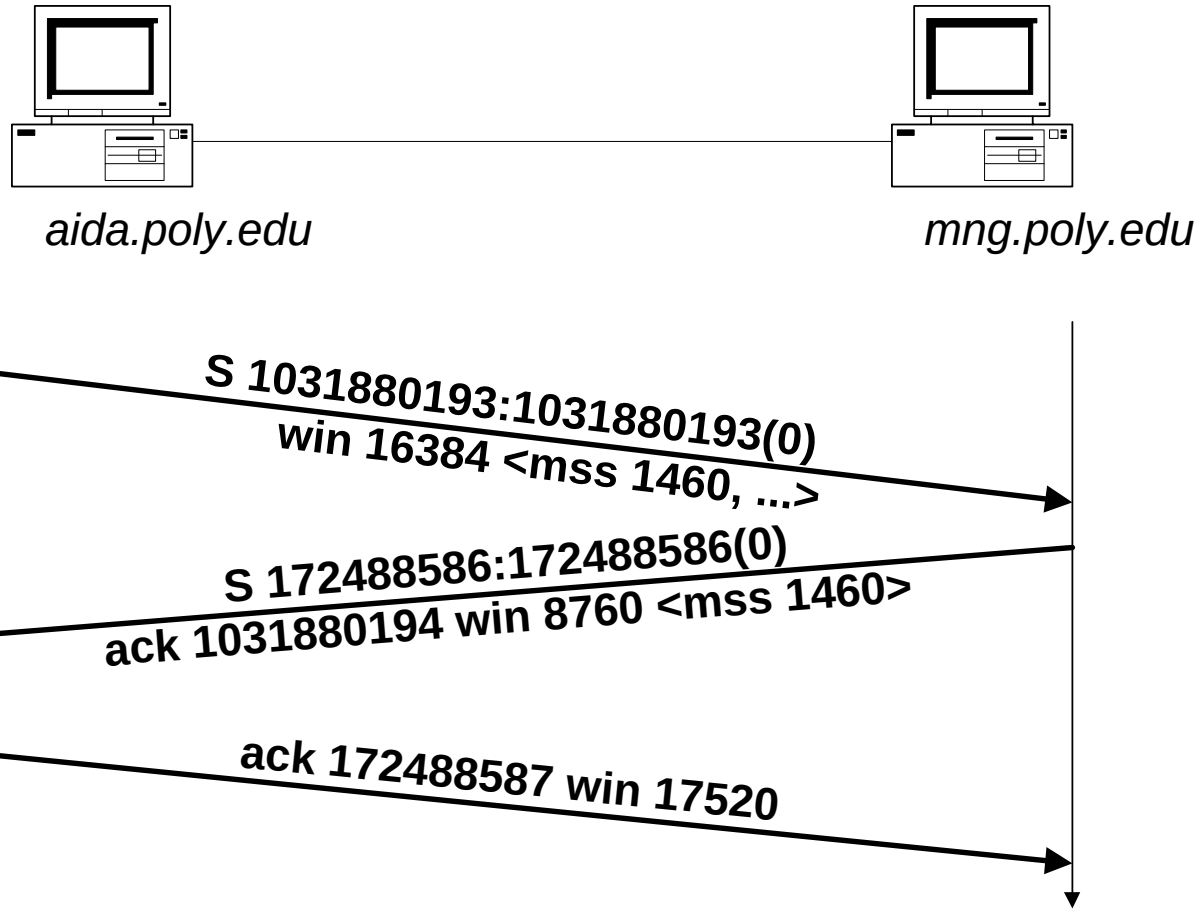
3 aida.poly.edu.1121 > mng.poly.edu.telnet: . ack 172488587 win 17520

4 aida.poly.edu.1121 > mng.poly.edu.telnet: P 1031880194:1031880218(24)
ack 172488587 win 17520

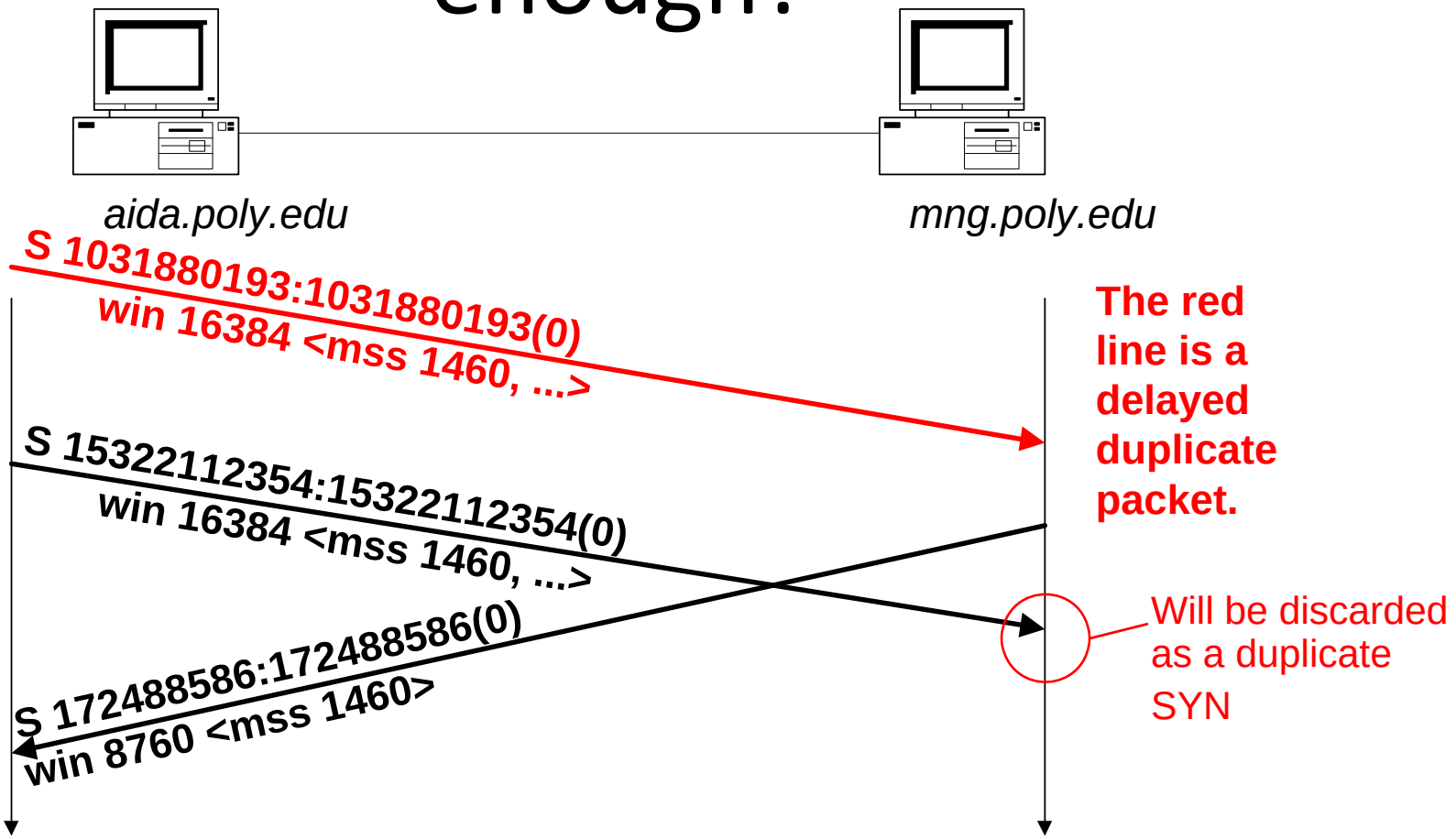
5 mng.poly.edu.telnet > aida.poly.edu.1121: P 172488587:172488590(3)
ack 1031880218 win 8736

6 aida.poly.edu.1121 > mng.poly.edu.telnet: P 1031880218:1031880221(3)
ack 172488590 win 17520

Three-Way Handshake



Why is a Two-Way Handshake not enough?



When aida initiates the data transfer (starting with SeqNo=15322112355), mng will reject all data.

TCP Connection Termination

- Each end of the data flow must be shut down independently **("half-close")**
- If one end is done it sends a FIN segment. This means that no more data will be sent

- Four steps involved:

  (1) X sends a FIN to Y **(active close)**

  (2) Y ACKs the FIN,

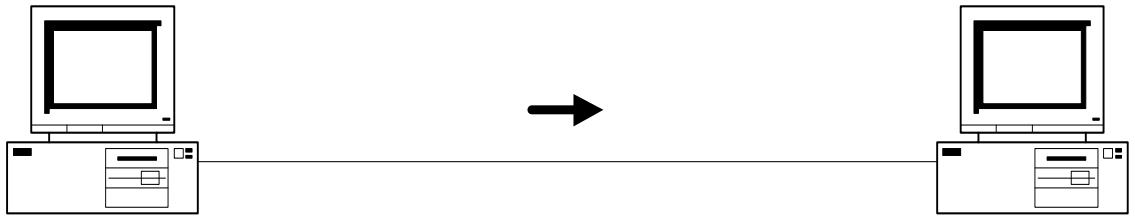
  (at this time: Y can still send data to X)

  (3) and Y sends a FIN to X **(passive close)**

  (4) X ACKs the FIN.

Connection termination with tcpdump

aida issues
an "telnet mng"



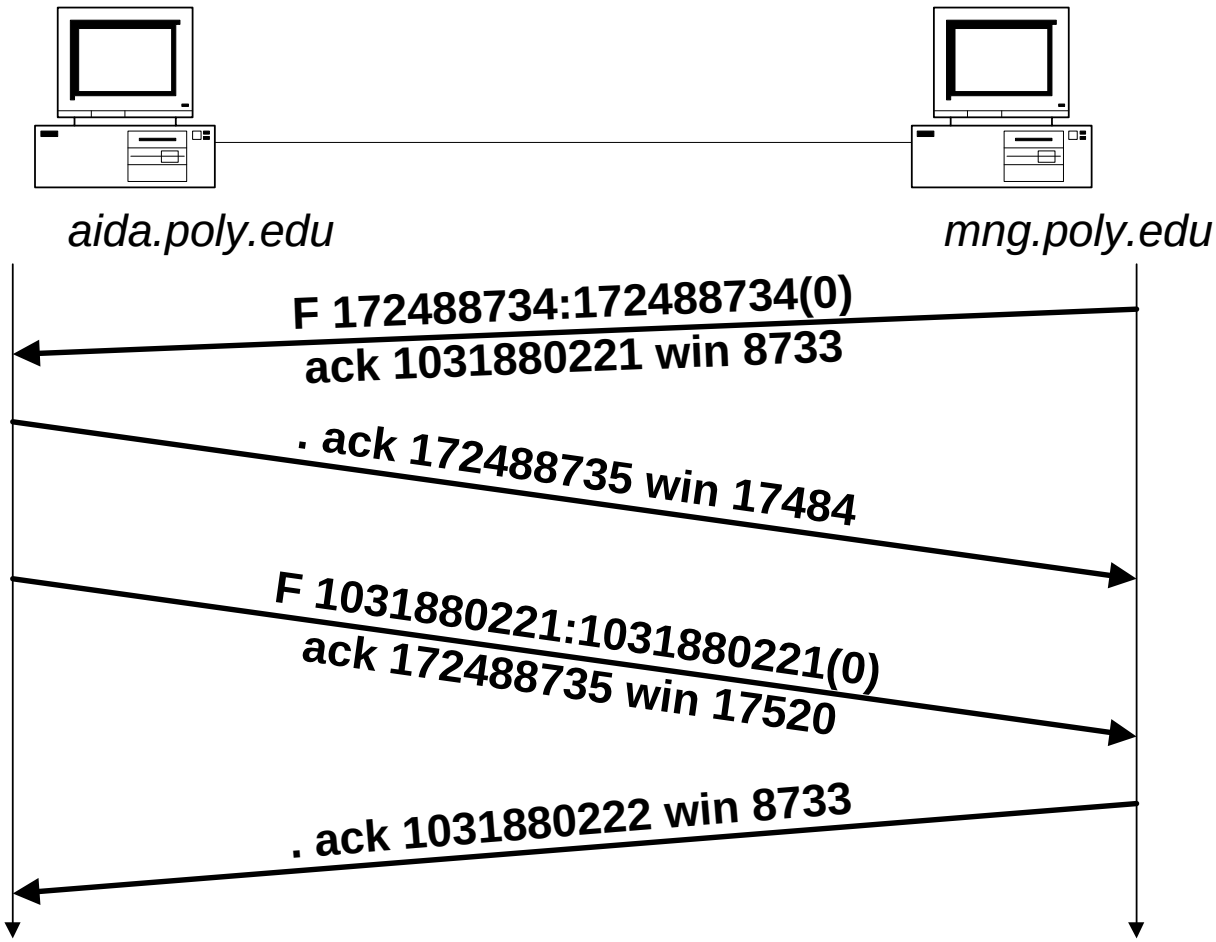
1 mng.poly.edu.telnet > aida.poly.edu.1121: F 172488734:172488734(0)
ack 1031880221 win 8733

2 aida.poly.edu.1121 > mng.poly.edu.telnet: . ack 172488735 win 17484

3 aida.poly.edu.1121 > mng.poly.edu.telnet: F 1031880221:1031880221(0)
ack 172488735 win 17520

4 mng.poly.edu.telnet > aida.poly.edu.1121: . ack 1031880222 win 8733

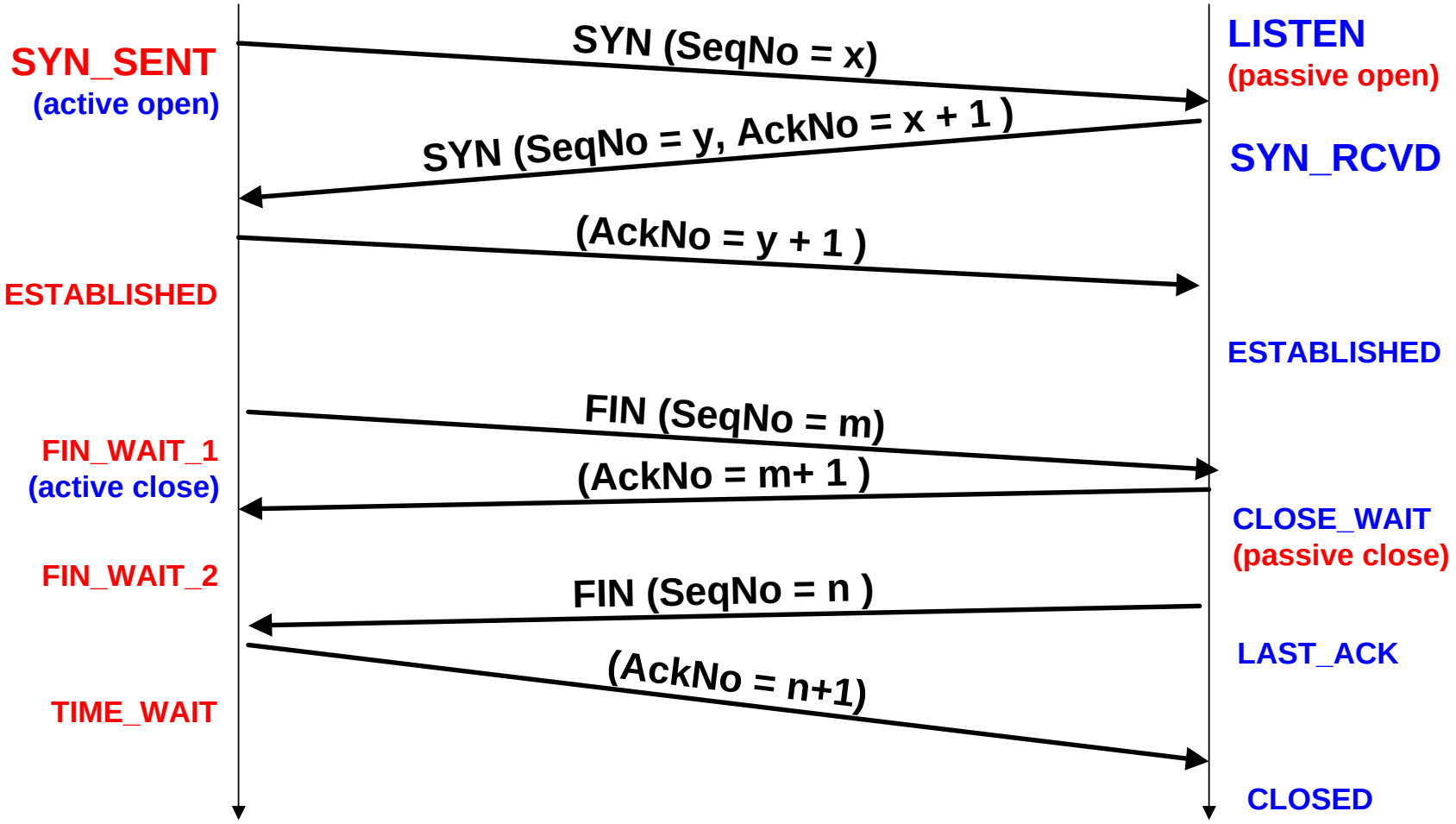
TCP Connection Termination



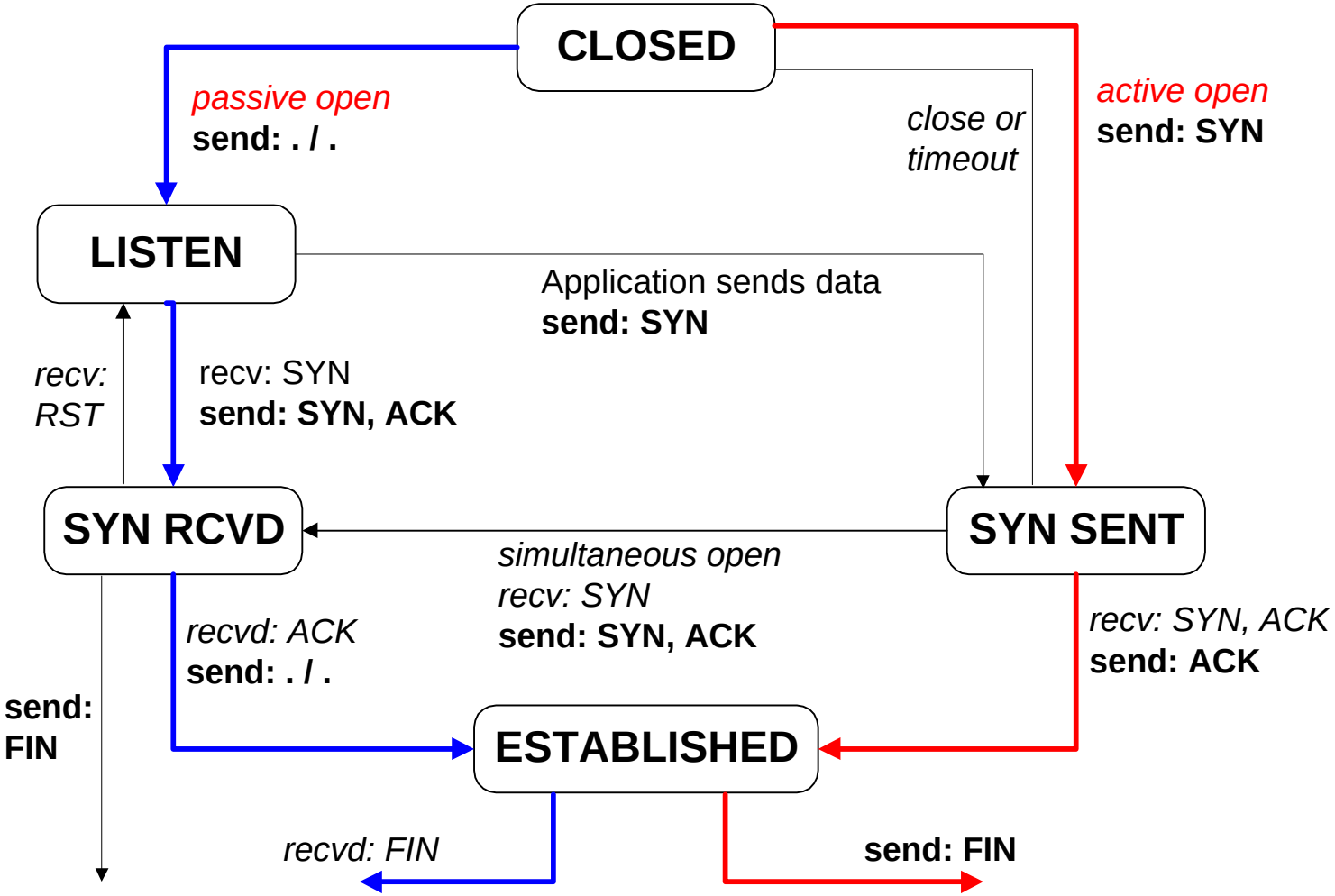
TCP States

State	Description
CLOSED	No connection is active or pending
LISTEN	The server is waiting for an incoming call
SYN RCVD	A connection request has arrived; wait for Ack
SYN SENT	The client has started to open a connection
ESTABLISHED	Normal data transfer state
FIN WAIT 1	Client has said it is finished
FIN WAIT 2	Server has agreed to release
TIMED WAIT	Wait for pending packets (“2MSL wait state”)
CLOSING	Both Sides have tried to close simultanesously
CLOSE WAIT	Server has initiated a release
LAST ACK	Wait for pending packets

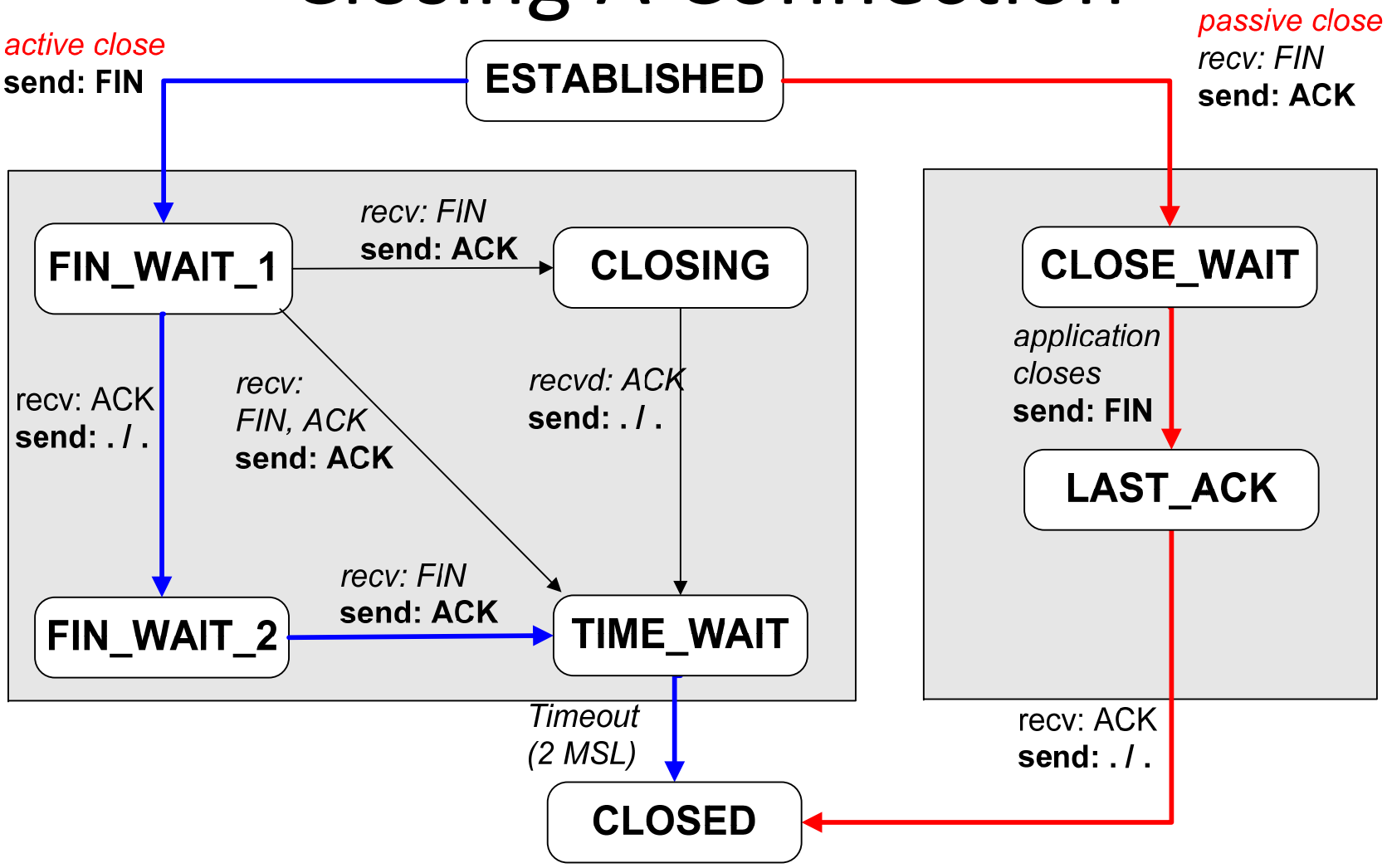
TCP States in “Normal” Connection Lifetime



TCP State Transition Diagram Opening A Connection



TCP State Transition Diagram Closing A Connection



2MSL Wait State

2MSL Wait State = TIME_WAIT

- When TCP does an active close, and sends the final ACK, the connection **must stay in in the TIME_WAIT state for twice the maximum segment lifetime**.

2MSL= 2 * Maximum Segment Lifetime

- Why?
  TCP is given a chance to resent the final ACK. (Server will timeout after sending the FIN segment and resend the FIN)
- The MSL is set to 2 minutes or 1 minute or 30 seconds.

Resetting Connections

- Resetting connections is done by setting the RST flag
- **When is the RST flag set?**
  - Connection request arrives and no server process is waiting on the destination port
  - Abort (Terminate) a connection
    Causes the receiver to throw away buffered data.
    Receiver does not acknowledge the RST segment

What is Flow/Congestion/Error Control ?

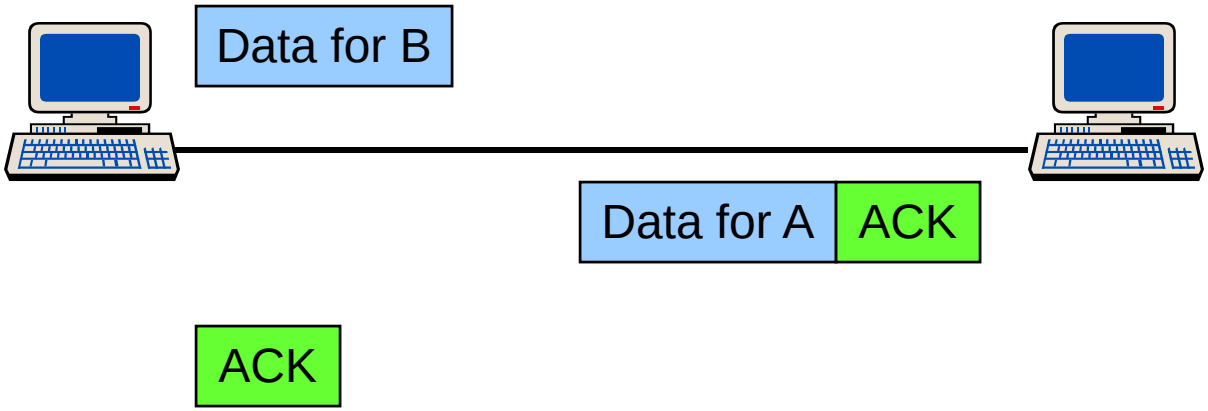
- **Flow Control:** Algorithms to prevent that the sender overruns the receiver with information
- **Error Control:** Algorithms to recover or conceal the effects from packet losses
- **Congestion Control:** Algorithms to prevent that the sender overloads the network

→ The goal of each of the control mechanisms are different.

→ In TCP, the implementation of these algorithms is combined

Acknowledgements in TCP

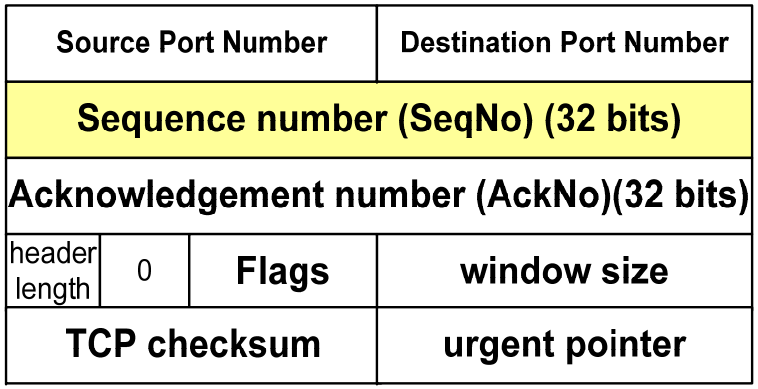
- TCP receivers use acknowledgments (ACKs) to confirm the receipt of data to the sender
- Acknowledgment can be added ("piggybacked") to a data segment that carries data in the opposite direction
- ACK information is included in the the TCP header
- Acknowledgements are used for flow control, error control, and congestion control



Sequence Numbers and Acknowledgments in TCP

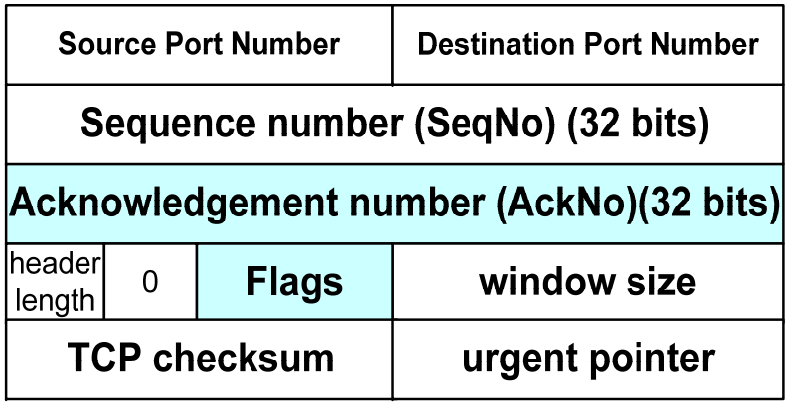
- TCP uses sequence numbers to keep track of transmitted and acknowledged data
- Each transmitted byte of payload data is associated with a sequence number
- **Sequence numbers count bytes and not segments**
- Sequence number of first byte in payload is written in *SeqNo* field
- Sequence numbers wrap when they reach 2^{32}-1

- The sequence number of the first sequence number (Initial sequence number) is negotiated during connection setup



Sequence Numbers and Acknowledgments in TCP

- An acknowledgment is a confirmation of delivery of data
- When a TCP receiver wants to acknowledge data, it
  - writes a sequence number in the AckNo field, and
  - sets the ACK flag

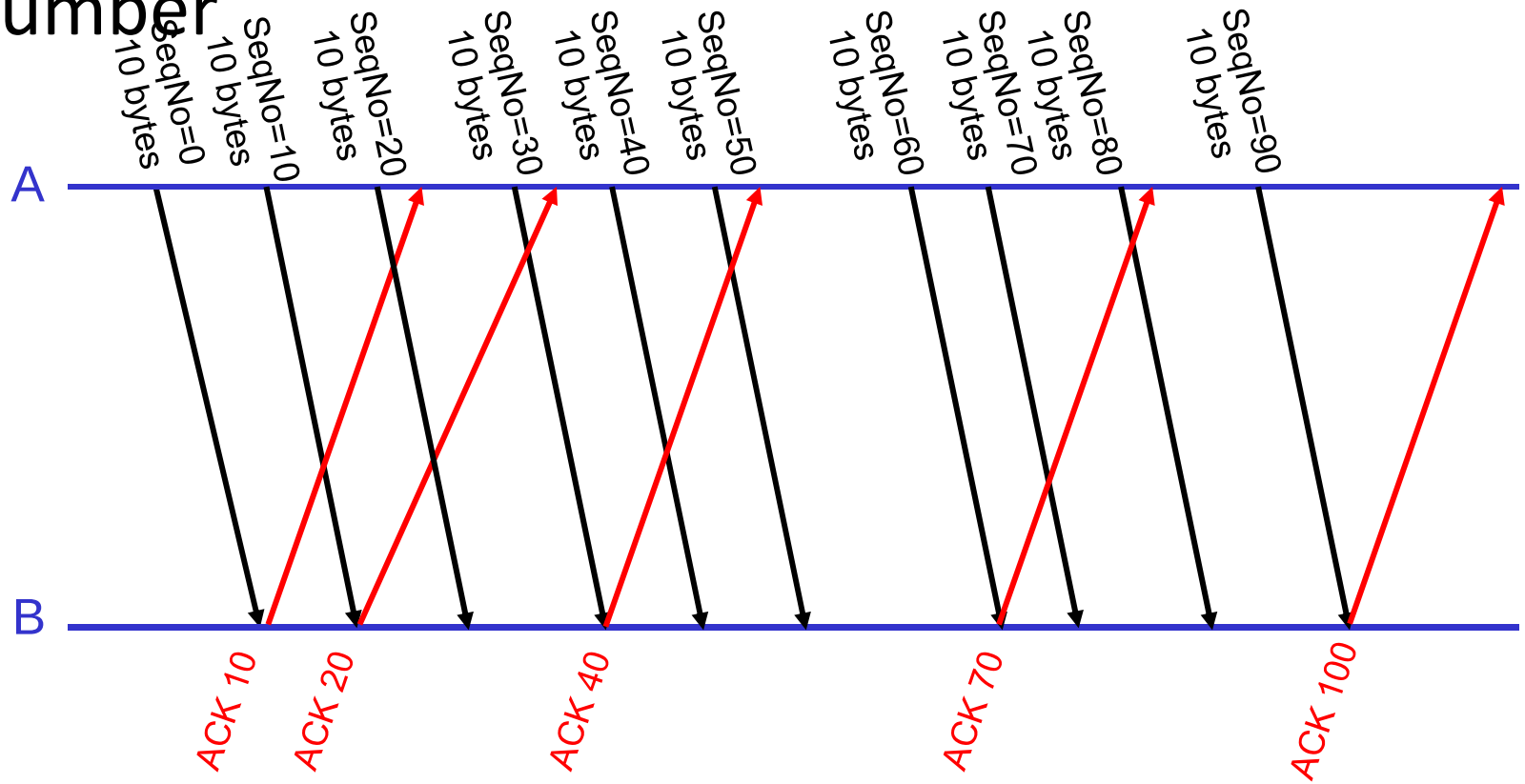


IMPORTANT: An acknowledgment confirms receipt for all unacknowledged data that has a smaller sequence number than given in the AckNo field

Example: AckNo=5 confirms delivery for 1,2,3,4 (but not 5).

Cumulative Acknowledgements

- TCP has **cumulative acknowledgements**:
  An acknowledgment confirms the receipt of all unacknowledged data with a smaller sequence number



Cumulative Acknowledgements

- With cumulative ACKs, the receiver can only acknowledge a segment if all previous segments have been received

- With cumulative ACKs, receiver cannot selectively acknowledge blocks of segments:
  e.g., ACK for S_0-S_3 and S_5-S_7 (but not for S_4)

- Note: The use of cumulative ACKs imposes constraints on the retransmission schemes:
  - In case of an error, the sender may need to retransmit all data that has not been acknowledged

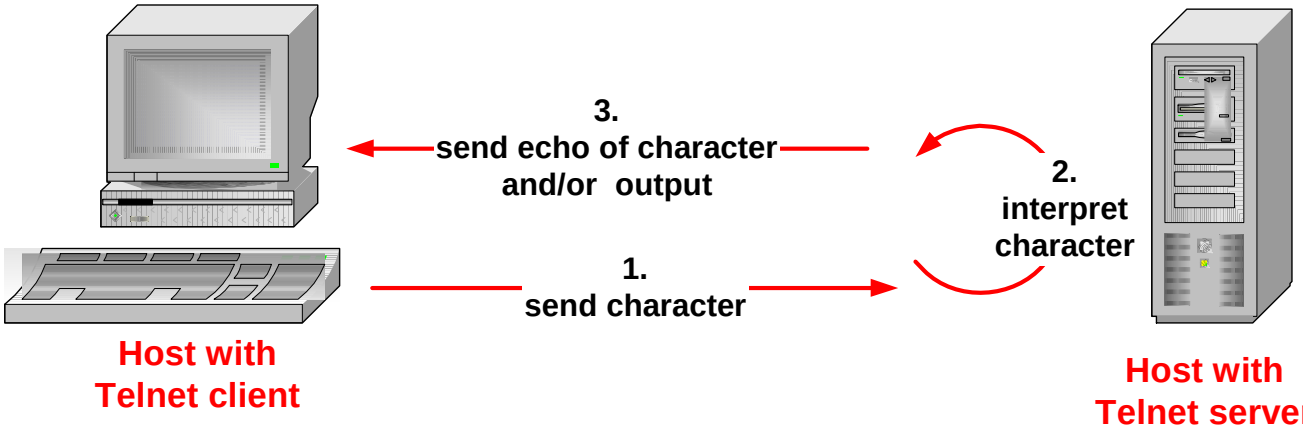
Rules for sending Acknowledgments

- TCP has rules that influence the transmission of acknowledgments

- Rule 1: Delayed Acknowledgments
  - *Goal:* Avoid sending ACK segments that do not carry data
  - *Implementation:* Delay the transmission of (some) ACKs

- Rule 2: Nagle's rule
  - *Goal:* Reduce transmission of small segments
    Implementation: A sender cannot send multiple segments with a 1-byte payload (i.e., it must wait for an ACK)

Observing Delayed Acknowledgements

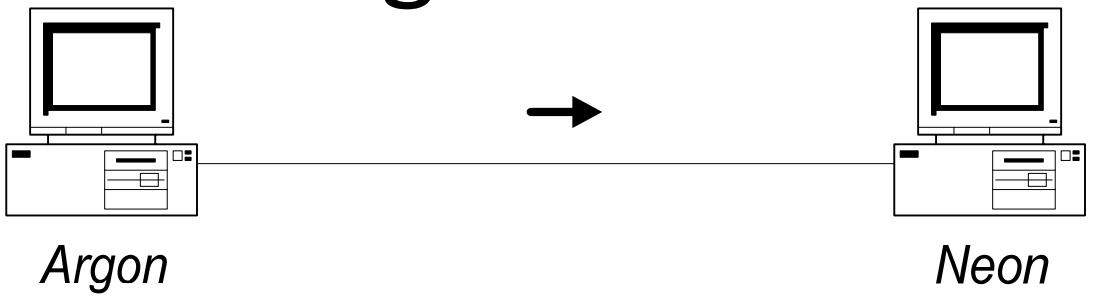


- Remote terminal applications (e.g., Telnet) send characters to a server. The server interprets the character and sends the output at the server to the client.

- For each character typed, you see three packets:
  1. **Client → Server:** Send typed character
  2. **Server → Client:** Echo of character (or user output) and acknowledgement for first packet
  3. **Client → Server:** Acknowledgement for second packet

Observing Delayed Acknowledgements

Telnet session from Argon to Neon



- This is the output of typing 3 (three) characters :

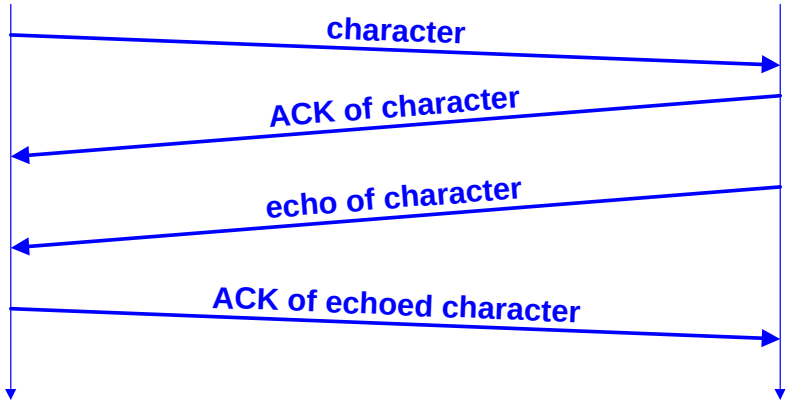
Time 44.062449: Argon → Neon: Push, SeqNo 0:1(1), AckNo 1
Time 44.063317: Neon → Argon: Push, SeqNo 1:2(1), AckNo 1
Time 44.182705: Argon → Neon: No Data, AckNo 2

Time 48.946471: Argon → Neon: Push, SeqNo 1:2(1), AckNo 2
Time 48.947326: Neon → Argon: Push, SeqNo 2:3(1), AckNo 2
Time 48.982786: Argon → Neon: No Data, AckNo 3

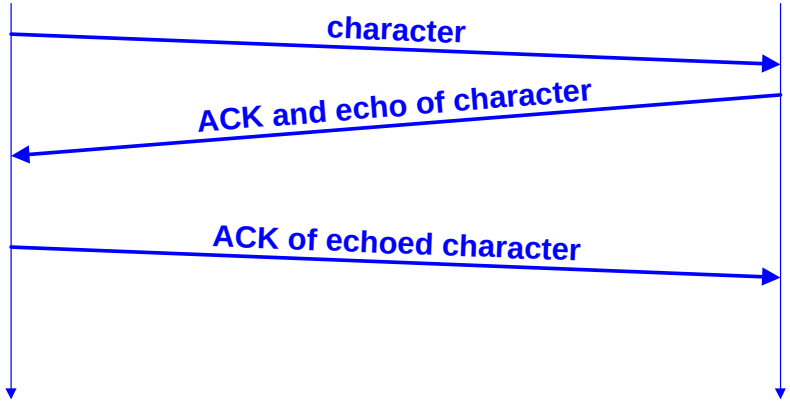
Time 55.116581: Argon → Neon: Push, SeqNo 2:3(1) AckNo 3
Time 55.117497: Neon → Argon: Push, SeqNo 3:4(1) AckNo 3
Time 55.183694: Argon → Neon: No Data, AckNo 4

Why 3 segments per character?

- We would expect four segments per character:



- But we only see three segments per character:



- This is due to delayed acknowledgements

Delayed Acknowledgement

- TCP delays transmission of ACKs for up to 200ms

- **Goal:** Avoid to send ACK packets that do not carry data.
  - The hope is that, within the delay, the receiver will have data ready to be sent to the receiver. Then, the ACK can be piggybacked with a data segment

  In Example:

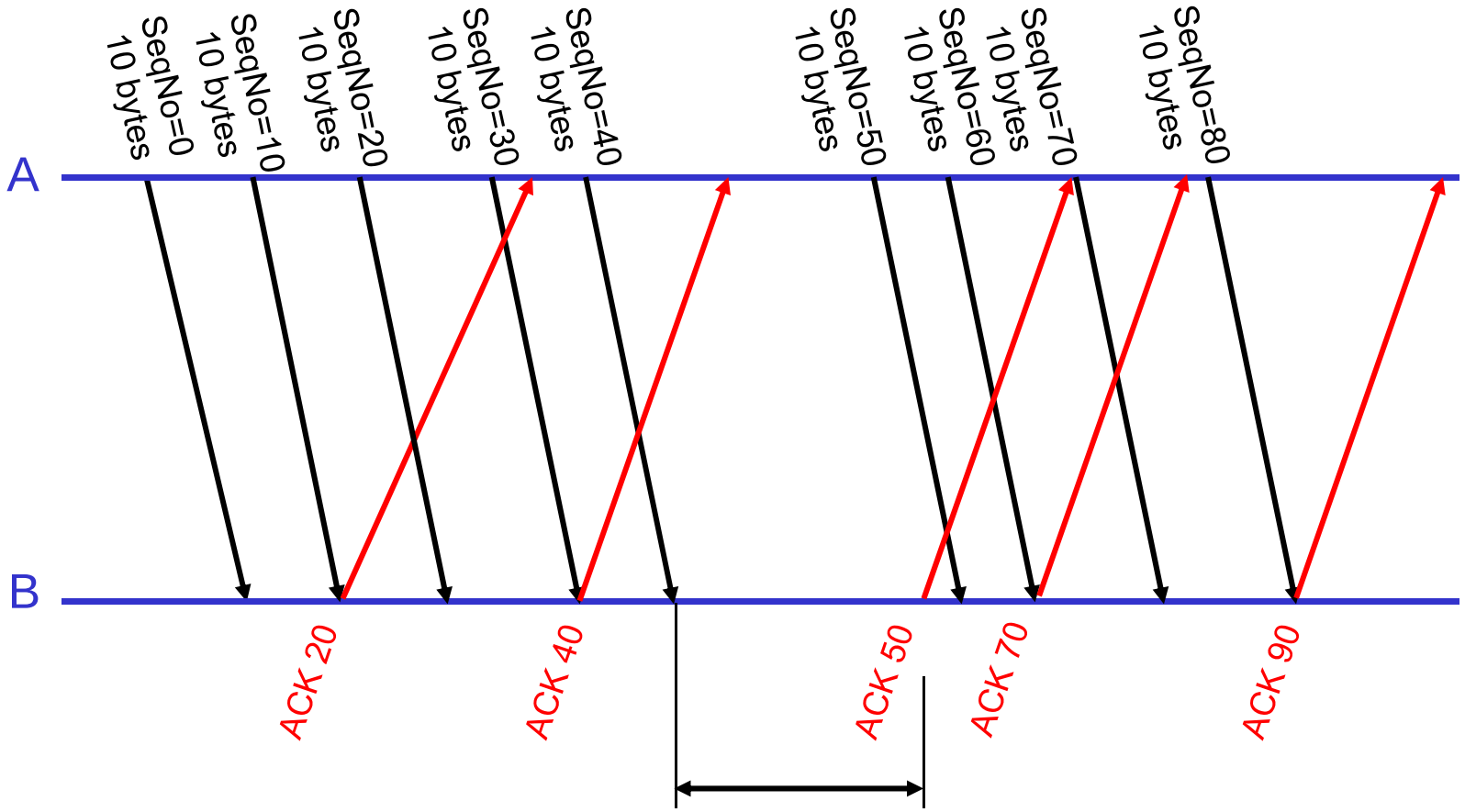
  - Delayed ACK explains why the "ACK of character" and the "echo of character" are sent in the same segment
  - The duration of delayed ACKs can be observed in the example when Argon sends ACKs

Exceptions:

- ACK should be sent for every second full sized segment
- Delayed ACK is not used when packets arrive out of order

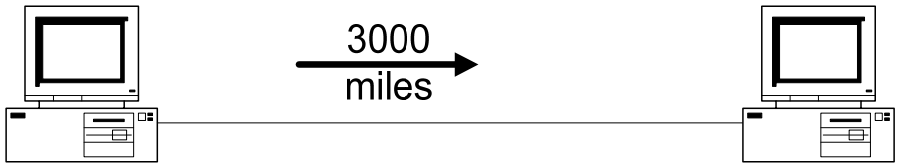
Delayed Acknowledgement

- Because of delayed ACKs, an ACK is often observed for every other segment



Observing Nagle's Rule

Telnet session
between argon.cs.virginia.edu
and
tenet.cs.berkeley.edu



- This is the output of typing 7 characters :

Time 16.401963: Argon → Tenet: Push, SeqNo 1:2(1), AckNo 2
Time 16.481929: Tenet → Argon: Push, SeqNo 2:3(1) , AckNo 2

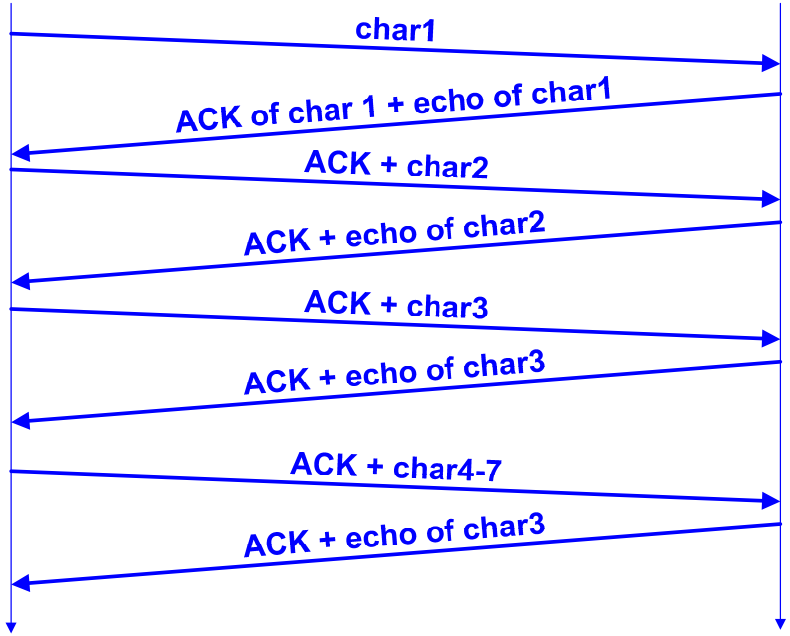
Time 16.482154: Argon → Tenet: Push, SeqNo 2:3(1) , AckNo 3
Time 16.559447: Tenet → Argon: Push, SeqNo 3:4(1), AckNo 3

Time 16.559684: Argon → Tenet: Push, SeqNo 3:4(1), AckNo 4
Time 16.640508: Tenet → Argon: Push, SeqNo 4:5(1) AckNo 4

Time 16.640761: Argon → Tenet: Push, SeqNo 4:8(4) AckNo 5
Time 16.728402: Tenet → Argon: Push, SeqNo 5:9(4) AckNo 8

Observing Nagle's Rule

- **Observation:** Transmission of segments follows a different pattern, i.e., there are only two segments per character typed



- Delayed acknowledgment does not kick in at Argon
- The reason is that there is always data at Argon ready to sent when the ACK arrives
- Why is Argon not sending the data (typed character) as soon as it is available?

Observing Nagle's Rule

- **Observations:**
  - Argon never has multiple unacknowledged segments outstanding
  - There are fewer transmissions than there are characters.

- This is due to **Nagle's Rule:**
  - Each TCP connection can have only one small (1-byte) segment outstanding that has not been acknowledged

- **Implementation:** Send one byte and buffer all subsequent bytes until acknowledgement is received.Then send all buffered bytes in a single segment. (Only enforced if byte is arriving from application one byte at a time)

- **Goal of Nagle's Rule:** Reduce the amount of small segments.
- The algorithm can be disabled.

Nagle's Rule

- Only one 1-byte segment can be in transmission (Here: Since no data is sent from B to A, we also see delayed ACKs)